RCA MICROSYSTEMS 1800

User Manual for the RCA Microboard Computer Development System (MCDS) CDP18S693 and CDP18S694

MPM-293A Sugges`ed Price $10.00

User Manual for the RCA Microboard Computer Development System (MCDS) CDP18S693 and CDP18S694

RCA Solid State

Brussels • Buenos Aires • Hamburg • Madrid • Mexico City • Milan
Montreal • Paris • Sao Paulo • Somerville NJ • Stockholm
Sunbury-on-Thames • Taipei • Tokyo

CLASS A
RADIO INTERFERENCE WARNING

This equipment complies with the requirements in Part 15 of FCC Rules for a Class A computing device. Operation of this equipment in a residential area may cause unacceptable interference to radio and TV reception requiring the operator to take whatever steps are necessary to correct the interference.

2488834-1

The software described in this manual is copyrighted by RCA Corporation.

Information furnished by RCA is believed to be accurate and reliable. However, no responsibility is assumed by RCA for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of RCA.

Copyright 1982 by RCA Corporation
(All rights reserved under Pan-American Copyright Convention)

Trademark(s)® Registered
Marca(s) Registrada(s)

Printed in USA/7-82

Foreword

The RCA COSMAC Microboard Computer Development Systems (MCDS) CDP18S693 and CDP18S694 are economical and versatile systems capable of facilitating the design and development of the hardware and software for applications based on the RCA 1800 series of CMOS microprocessor products. With the optional Run-time BASIC available on ROM, and with the addition, if needed, of any of the many available expansion Microboards, the MCDS may be used very effectively for control, testing, or other dedicated microcomputer applications.

The CDP18S693 includes an eight-card chassis with case, a 5-volt power supply, a CDP18S601 Microboard Computer, a CDP18S652 Microboard Combination Memory and Tape I/O Control Module augmented with a ROM-based monitor program and a ROM-based extended BASIC3 interpreter, an audio cassette magnetic tape system for memory storage, and the cables needed for connecting a data terminal and for connecting the cassette drive system to the CDP18S652.

The 2-kilobyte monitor program UT62 enables the user to modify and examine memory, debug programs, store and retrieve data on tape, and start execution of programs. The BASIC3 interpreter (12 kilobytes) features full floating-point arithmetic, line editing capability, and "trace" debugging for program creation. The audio cassette tape system provides for economical storage and retrieval of programs or data.

The CDP18S694 has all the features of the CDP18S693 plus the following. In an additional three-ROM set on the CDP18S652, a text editor and Level I assembler are provided that enable the user to create CDP1802 machine language programs in Level I mnemonic language.

A PROM Programmer Module is also provided along with a control program on cassette tape that enables the user to program a wide variety of EPROM's. A second audio cassette drive is included to support the editor and assembler operations.

This Manual describes in detail the hardware structure and the software features and commands of the Microboard Computer Development Systems (MCDS) CDP18S693 and CDP18S694. For a description of BASIC3, the user should refer to the manual **BASIC3 High-Level-Language Interpreter CDP18S841 User Manual,** MPM-841. The user should also refer to the **User Manual for the CDP1802 COSMAC Microprocessor,** MPM-201, for a detailed description of the instruction set and the architecture of the CDP1802 COSMAC Microprocessor.

Contents

Page

1. Setting Up the Systems 5
2. Features of the UT62 Monitor Program 7
   - UT62 Commands 7
   - ASCII Coding 9
   - Read and Type Routines 9
   - Examples of UT62 Usage 12
   - Tape READ and WRITE Subroutines 14
3. Running BASIC3 16
4. Running with MCDS COSMAC Resident Editor (CRE) 17
   - Introduction 17
   - CRE Operating Considerations 18
   - CRE Command Operation 20
   - CRE Commands 21
   - Using CRE 25
5. Running the MCDS COSMAC Resident Assembler (CRA) 28
   - Using the Assembler 29
   - The Location Counter 29
   - The Symbol Table 29
   - Expression Evaluation 29
   - COSMAC Level I Assembly Language ... 30
6. Hardware Structure and System Signals 37
   - Module Descriptions 37
   - System Signals 39
7. Memory Addressing and Expansion 41
   - On-Board Memory Addressing 41
   - Memory Expansion 42
8. Input/Output Interfacing 44
   - Two-Level I/O Interfacing 44
   - Serial I/O Interfacing 44
   - Parallel I/O Interfacing 45
   - I/O Expansion 46
   - System Signals in I/O Interfacing 47

Page

9. Using the Cassette Tape Drive 49
   - Operation 49
   - System Care 49
   - Controls 49
10. Using the PROM Programmer 51
    - Installation 51
    - Memory Organization and Use 51
    - Software Considerations 52
    - Precautions 53
    - Operations and Commands 54
    - Examples of Compound Operations 58

Appendix A— Specifications for the Microboard Computer Development Systems (MCDS) CDP18S693 and CDP18S694 59

Appendix B— UT62 Monitor Program Listing .. 60

Appendix C - Tape Handling Precautions 87

Appendix D - Connection List for Terminal Interface Cable CDP18S516 ... 88

Appendix E— Instruction Summary for RCA CDP1802A, CDP1804, and CDP1805 COSMAC Microprocessors 89

Appendix F— ASCII-Hex Table 93

Appendix G— Logic Diagram, Layout Diagram and Parts List for CDP18S652 Microboard Combination Memory and Tape I/O Control Module 94

Appendix H— Hardware Description for PROM Programmer Module 97

Appendix I— PROM Programmer Specifications 103

Appendix J— Generating Hard Copy on MCDS 104

Index 105

1. Setting Up the Systems

The CDP18S693 and CDP18S694 Microboard Computer Development Systems, as supplied, need only be cabled to the user's data terminal and connected as shown in Fig. 1 to become operational. The data terminal may be one having either a current loop interface or an EIA RS232C interface operating at 110, 300, or 1200 baud. In the latter case, the additional negative voltage required is supplied by the user.

a wire. These additions provide assurance that the EIA RS232C circuitry is properly biased for operation. It should also be noted that the EIA RS232C terminal interface can be used to advantage with the RCA Interactive Data Terminal VP-3301 which provides an NTSC-compatible video output for direct use with a 525-line color or black-and-white video monitor. When the VP3301 is used, it is permissible to connect 21 and 11 on the Universal Backplane and not use a

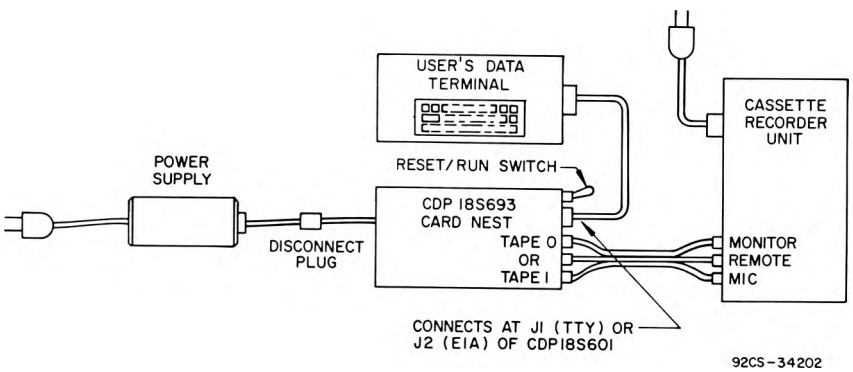


Fig. 1—CDP18S693 Microboard Computer Development System configuration

A cable CDP18S516 is provided for connecting the user's data terminal to the system. The cable is labeled EIA and is keyed at the 10-pin connector end (P1) to plug into J2 (EIA) of the CDP18S601. The other end is a standard EIA RS232C 25-pin delta connector that mates with the connector on the data terminal. No handshaking lines are required for operation. The details of the cable wiring are given in Appendix D, Connection List for Terminal Interface Cable CDP18S516.

When an EIA RS232C terminal interface is used, it should be noted that (1) the +5 volts dc supply needed is available at pins 21 and Y of the Universal Backplane (See Table IX), (2) the user must supply a negative voltage of −5 to −15 volts at pin 11 of the Backplane, and (3) pin 20 (+V) and pin 21 should be jumpered with negative voltage for the RS232C interface. This arrangement allows operation with a single voltage.

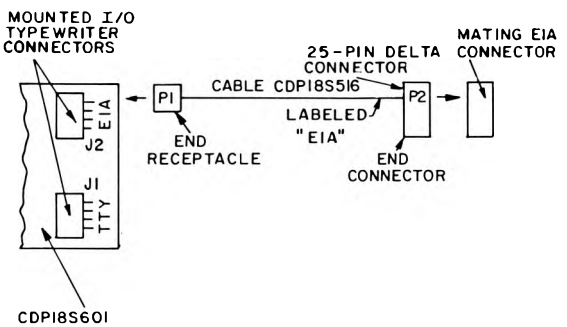


Fig. 2—Cable connection for I/O data terminals

The cassette tape interface cable connects the CDP18S652 controller board to the cassette drive mechanism. Wire markers identify plugs "MIC," "REM," and "MON" at each end of the cable assembly. The "REM" plug at each end of the cable is smaller than the other plugs and is to be plugged into the "REMOTE" jack at the tape recorder end. Plug "MON" is plugged into the "MONITOR" jack and plug "MIC" into the "MIC" jack. The other end of the cable is then plugged into the corresponding jacks of the tape controller board CDP18S652 at the tape 0 or the tape 1 interface. The cassette cable can be attached or removed with power on the system.

To start the system, the user should first make sure that the various units are connected as shown in Fig. 1, that the volume control on the cassette recorder unit is set at maximum (10), and that the tone control is set at maximum (10). The user should then place the RESET/RUN switch in the reset position and plug in the power supply and the cassette tape system power cable. Next, place the RESET/RUN switch in the run position. Press the carriage return key on the terminal and the system should respond with an asterisk (*) indicating that the system is ready to accept user commands. If an asterisk prompt is not received, the user should examine all connections and restart the system with the RESET/RUN switch and then press the carriage return key.

The following brief tests of the system operation utilize the Insert (I) and Display (D) instructions of the monitor program and indicate whether the monitor program is operational. The carriage return key (CR) signifies the "end of transmission" to the system when it is waiting for a specific response. (H indicates hexadecimal notation)

Test 1—Display

Type:
D8000 4 (CR)..Display 4 bytes at memory 8000H

System Response:
8000 7100 F880

Test 2—Insert and Display

Type:
I0000 11223344(CR) ..Insert bytes 11, 22, 33
..44 into memory at 0000H

System Response:
*

Type:
D0000-0003(CR) ..Display memory contents from
..0000H through 0003H

System Response:
0000 11223344

The Insert and Display instructions just illustrated are discussed in detail in the next section along with other features of the monitor program.

2. Features of UT62 Monitor Program

The purpose of the monitor program is to provide a convenient place for the microprocessor to begin running. The monitor allows the user to start BASIC3, the editor, the assembler, or a user-generated program at any address.

While the microprocessor is running in the UT62 monitor, several memory functions are available through the data terminal. On a byte or block basis, memory can be displayed, modified, or moved. Also, the microprocessor register values on entry into the monitor program can be examined by displaying memory at location 8C00H. This REGISTER SAVE feature can be used to debug machine language programs by inserting an IDLE instruction (00) in the user's code at the appropriate place and then running the monitor program by means of the RESET/RUN switch. All registers are saved in memory at 8C00H except the values of R0, R1, and R4.1 which are lost in the process.

The monitor can also load and save programs on the cassette tape units and display ASCII files from tape on to the terminal. The tape SAVE routine can store 1 byte or any contiguous block of bytes from any space in memory. The tape LOADER will restore any saved memory and can also load tapes created by the Level 1 assembler of the CDP18S694.

To start running the monitor program, the user should place the RESET/RUN switch in the reset position and then move it back to the run position. The user should then press the carriage return key (CR) on the terminal. The * prompt signifies that the system has calculated the baud rate of the terminal and is now ready to accept monitor commands.

UT62 Commands

Following is a description of the thirteen UT62 commands. Note that all address, data, and byte counts are entered as hexadecimal numbers (indicated by the letter H following the number). In these examples, the characters generated by the system are underlined.

D Commands

Name: Memory Display

Purpose: To allow a specified area of memory to be displayed on the user terminal.

Format: **D(START ADDR)(OPTION)(CR)**

Action: The contents of memory, beginning at the specified (START ADDR) will be transmitted to the user terminal. (OPTION) allows the transmission of either a specific number of bytes preceded by a space or an inclusive address range preceded by a hyphen. If the option is not specified, a default value of 1 byte results.

Examples: **D02F8 8(CR)**
D02F8-02FF(CR)
Both of these examples produce the same output.

I Commands

Name: Memory Insert

Purpose: To alter the contents of memory beginning at the specified address.

Format: **I (START ADDR)(SPACE)**
(DATA)[(CONT)](CR)

Action: A memory location is accessed at the specified (START ADDR). The (DATA) required is one byte specified by two hex digits. The (CONT) option allows data to be continued changing the current memory address. If a (SEMICOLON) is entered and after a user-inserted (CR)(LF), a new address is anticipated. The semicolon allows non-contiguous memory to be loaded with a single insert command. The command may be terminated at any point by the entry of a (CR) or (SEMICOLON).

Examples: **I02F8 7100F840B0F88CB1(CR)**
I02F8 7100F840B0;(CR)(LF)
03B6 94FB903A0F(CR)
The second example enters data into two memory areas, starting at 02F8 and 03B6.

M Commands

Name: Memory Move

Purpose: To move a block of data from one area of memory to another area.

Format: **M(SOURCE ADDR)(OPTION)(SPACE)**
(DEST ADDR)(CR)

Action: Data is copied from memory location beginning at the (SOURCE ADDR) into locations specified by the (DEST ADDR). (OPTION) allows the transfer of either a specific number of bytes preceded by a space or an inclusive address range preceded by a hyphen. There is no restriction on the direction of the move and the areas may overlap.

Examples: **M02F8 8 03F8(CR)**
M02F8-02FF 03F8(CR)

M03B0-03BF 02B0(CR)
M03B0-03BF 03B2(CR)

F Commands
Name: Memory Fill
Purpose: To load a defined area of memory with a specific constant.
Format: **F(START ADDR)(OPTION)(SPACE)(DATA)(CR)**
Action: The specified (DATA) is loaded into memory beginning at the (START ADDR). (OPTION) allows the loading of either a specific number of bytes preceded by a space or an inclusive address range preceded by a hyphen.
Examples: **F02F8 8 00(CR)**
F02F8-02FF 00(CR)
These examples fill with zeros the eight bytes beginning at location 42F8.

S Commands
Name: Memory Substitute
Purpose: To display and, if desired, alter the contents of sequential memory locations beginning at the specified address.
Format: **S(START ADDR)(OPTION)(CR)**
Action: A memory location is accessed at the specified (START ADDR). Its contents will not be displayed, however, until (OPTIONS) is entered. (OPTIONS) allows two methods of display. If (SPACE) is entered, the current data will be displayed on the same line followed by a hyphen. New data may be entered at this point. Only the last byte entered will be written. If no data is entered, the current data will remain unchanged. If a (LF) is entered, a (CR)(LF) will result and the current memory address will be echoed to the terminal prior to the printing of the current data. New data may be entered as described above. The command can be terminated by a (CR) or continued by the entry of any of the (OPTIONS).

Examples: **S02F8 63-71 00- 0F-C0(CR)**
The current data of 63 is changed to 71. The 00 data is retained, and the 0F is changed to C0.
S02F8 71- 00- C0- 11-82(LF)
02FC 52-AE(LF)
02FD 00-F8 11-40 23-A3(CR)
In this example, the 71, 00, and C0 are retained and the 11 is changed to 82. Each (LF) causes the next address to be typed followed by its data.

P Commands
Name: Program Run
Purpose: To allow a user program to be run beginning at the specified address.
Format: **P[(START ADDR)](CR)**
Action: The user program will begin execution at the specified (START ADDR) with P = 0 and X = 0. If the (START ADDR) is not specified, the default value is 0000.

R Commands
Name: Read Tape
Purpose: To load a user's program from tape.
Format: **R**
Action: Load selected tape into memory. Tape specifies target address. Assembled or SAVED programs can be loaded.
Example†: **R**
TAPE# 1 (CR)
LOADING
*

W Commands
Name: Write Tape
Purpose: To save a user's program on tape.
Format: **W**
Action: Write user memory to tape from address AAAA to BBBB on tape selected.
Example†: **W**
FROM? 0100(CR)
TO? 0300(CR)
TAPE# 0(CR)
DONE!
*

J Commands
Name: Rewind Tapes
Purpose: To allow user to rewind tapes normally under system control.
Format: **J**
Action: Turn on tape drives and wait for user to press a key.
Example†: **J**
REWIND TAPES, THEN TYPE ANY KEY (X)
*

B Commands
Name: Run
Purpose: To start execution of BASIC3.
Format: **B**
Action: System prints out BASIC3 title and asks for "cold or warm" entry.
Examples†: **B**
BASIC3 VX.X
C/W?
CREADY
:

E Commands (CDP18S694 only)
Name: Run Editor
Purposes: To start execution of the text editor.
Format: **E**
Action: System pauses to search for top of free RAM and then issues the title and prompt.

†In these examples the system response is indicated by an underline.

Examples†: **E**
MCDS TAPE EDITOR VERSION X.X
=>

A Commands (CDP18S694 only)

Name: Run Assembler
Purpose: To start execution of the Level I assembler
Format: **A**
Action: System pauses to search for the top of the free RAM and then issues the title and starts by asking for "READ" tape. For subsequent action refer to Chapter 5, **Running the MCDS COSMAC Resident Assembler (CRA).**
Examples†: **A**
MCDS ASSEMBLER VER. X.X
READ?

C Commands

Name: Copy to Monitor
Purpose: To display ASCII file on the Video Monitor from tape.
Format: **C**
Action: Data in ASCII format is read from tape and displayed on the Video Monitor. After twenty Carriage Returns (ASCII 0DH). The display is stopped and 4 *'s are displayed on the bottom line. To restart the display, press any key. While characters are being written on the screen (not while the tape is being read) pressing the BREAK key will halt the display. Pressing the Q key will abort the routine and return the prompt. Pressing any other key will start the display. (The space bar is recommended because whichever key is pressed is echoed to the screen.)
Example†: **C**
TAPE# **0 (CR)**
user's code

ASCII Coding

The CDP18S693 and CDP18S694 are designed to interface to a data terminal via a serial ASCII code using either a 20-mA current loop or an EIA RS232C standard electrical interface. When a key is struck on a TTY terminal, the information denoting that character is converted to its ASCII code and appears on the output terminals as a serial data-bit stream. The serial data originating at the TTY for the letter 'M' is shown in Fig. 3. The character is framed by a start bit B and two stop bits FF. By convention two stop bits are used for data transmission at 10 characters per second although 1, 1½, or 2 are also acceptable outputs from various different terminals. A parity bit P is also shown. For even parity, the parity bit would be a '1' only if the 7 data bits contain an odd number of '1's. Hence, the total number of 1's in the eight intelligence bits is always an even number. Some data terminals may be set up to generate either even or odd parity. UT62 ignores the parity bit, so either even or odd parity is acceptable.

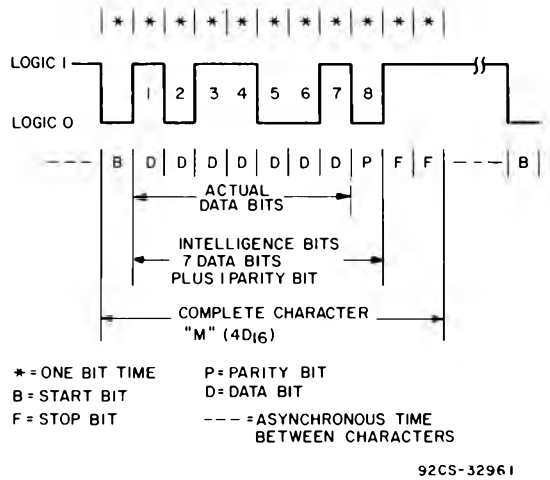

LOGIC 1
LOGIC 0
1 2 3 4 5 6 7 8
B D D D D D D P F F B
ACTUAL DATA BITS
INTELLIGENCE BITS 7 DATA BITS PLUS 1 PARITY BIT
COMPLETE CHARACTER "M" (4D16)
* = ONE BIT TIME
B = START BIT
F = STOP BIT
P = PARITY BIT
D = DATA BIT
--- = ASYNCHRONOUS TIME BETWEEN CHARACTERS
92CS-32961

Fig. 3—Data terminal bit serial output for the character "M".

Data from the MCDS are generated with the same format; i.e., a start bit, 7 data bits, a parity bit, and two stop bits. Note that the MCDS does not generate parity—the parity bit is always a '1' regardless of the data bits. Therefore, terminals interfacing to the MCDS should ignore the parity bit.

Read and Type Routines

The UT62 read and type routines provide the basic software mechanism for communication between the System and the data terminal. Several different routines are available to facilitate different types of I/O data transfers. These routines are listed in Table I.

These routines are designed to allow adoption to various terminal speeds and to determine whether or not characters read in should be "echoed" (i.e., typed back immediately). For these purposes, a 'sub-subroutine' called DELAY1 is included which provides the necessary bit timing delays to the read and type routines. DELAY1 uses the CDP1802 register RC as its program counter, which must be set-up to point to location 80EF. UT62 does this set up automatically when it is started. Any user program using a read or type routine must not alter RC, or must restore it to 80EF before calling a read or type routine. Also, the upper half of register RE (RE.1) contains a control constant. The least significant bit specifies echo (0 denotes echo, 1 denotes no echo). For full-duplex operation, then, this bit is a zero. Again, this bit is automatically set when UT62 is started and the CR or LF characters received.

The remainder of RE.1 constitutes a timing

†In these examples the system response is indicated by an underline.

parameter (TP). TP is calculated as follows:

$$TP = 2 \times \frac{\text{interval between two serial bits}}{320 \times (\text{CPU clock period})} \dagger$$

where the fraction is rounded to the nearest integer. For example, because a Teletypewriter Model 33 operates at 10 characters per second and 11 bits per character, for the CDP18S693 running from the supplied 2.0-MHz clock,

$$TP = 2 \times \left[\frac{\frac{1\text{s}}{10\text{ char}} \times \frac{1\text{ char}}{11\text{ bits}}}{320 \times \frac{1\text{ s}}{2.0 \times 10^6}} \right]$$

$$= 2 \times 56.8 \text{ (rounded to 57)}$$
$$= 114 \quad = 72H$$

Because for proper operation TP must be less than 255, there is a bound on the speed of terminals supported at any given clock rate. Faster terminals or slower clocks can be supported to the extent that roundoff errors do not cause bad timing. For example, at 2.0 MHz and 30 10-bit characters per second,

$$TP = 2 \times \left[\frac{\frac{1}{30} \times \frac{1}{10}}{320/2.0 \times 10^6} \right] = 2(20.8) = 42 \quad = 2AH$$

and the round-off error is small (2100 instead of 20.8). On the other hand, at 2.0 MHz with baud rates above 1200, the round-off error would be too high.

The utility program UT62 uses a subroutine "TIMALC" to generate the operating time constant, using the first character typed in by a user. This routine times the intervals between incoming bits to calculate TP and reads one bit to determine whether or not to echo. Specifically, if a CR is entered while TIMALC is running, then echoes will be provided; an LF suppresses echoes. In either case, RE. 1 is loaded with the appropriate constant. TIMALC also loads the subroutine pointer for the DELAY1 routine. The user of TYPE and READ has the option of calling TIMALC or setting up RE. 1 and the pointer to the DELAY1 routine himself. As a convenience to the user, UT62 leaves RE.1 and RC properly adjusted while performing a P operation and may be used unless they have been altered by the user. A summary of UT62 register utilization is given in Table II.

All read and type routines and TIMALC use R3 as their **program counter** and **return** to the caller with **SEP R5**. They can be called directly from a program that can use R5 as its program counter, or they may be called through the Standard Call and Return Technique (SCRT) described in the **User Manual for the CDP1802 COSMAC Microprocessor, MPM-201** in the Section **Programming Techniques** under "Subroutine Techniques". This programming technique is the most general and is recommended.

†The factor of 2 comes from the fact that the input serial waveform is sampled over two successive bit times. The factor of 320 comes from the fact that the time between samples is 20 instruction times, with each instruction taking 16 clock periods.

RE.1 is reserved for the operating constant (control constant 0 or 1 added to the timing parameter TP) discussed above.

RF.1 is used in certain cases to pass the byte being read or typed between the calling routine and these subroutines. When READ is exited, it leaves the input byte in RF.1. When TYPE is entered at location 81A4, the byte to be typed is taken from RF.1.

All routines alter RE.0 and RF.0. They also alter D, DF, and X. All TYPE routines modify RD.0 also. The READ routine leaves the input byte in D as well as in RF.1, but the byte in D will be destroyed if the Standard Call and Return Technique is used.

When TIMALC exits, R3.1 is left holding A.1 (READ) = A.1 (TYPE) = 81, but R3.0 is meaningless. When READ exits, R3 is ready for entry at READAH (see Table 1). When TYPE exits, R3 is ready for entry at TYPE5 (see same table). When DELAY exits, RC is ready for another call to DELAY1. When the Standard Call and Return Technique is used, R3 is automatically set up.

The **READ routine** has two entry points - READ and READAH. The former acts as described above and has no other side-effects. The latter operates just as READ does, but with the following side-effect. If the character read in is a hex character (0-9, A-F) then the 16-bit contents of RD are shifted four bits to the left, and the 4-bit hex equivalent of the input character is entered at the right. DF is then set to 1 on exiting. If the input character is not a hex character, RD is not affected, but DF is set to 0 on exiting.

CAUTION: A READ may immediately be followed by another READ, but not immediately by a TYPE. The caller should wait 1.5 bit times first, which he can do by entering TYPE at TYPED or TYPE5D or by calling DELAY1, with a parameter of 7 or greater.

The DELAY1 subroutine assumes that the calling program counter is R3. It uses the value, n, of the immediate byte at M(R3) to generate a delay equal to

$$(20 + m\,(2n + 6)) \text{ instruction times}$$

where m is time constant in RE.1 (see previous discussion). It then increments R3 past the calling parameter and returns via a SEP R3.

The **TYPE routine** has six different entry points. Three of them simply specify different places to fetch the character from: TYPE types from RF.1, TYPE5 types from M(R5) and increments R5, and TYPE6 types from M(R6) and increments R6. TYPED and TYPE5D are entries that provide a 1.5-bit delay before going to TYPE and TYPE5, respectively. The purpose of this delay is to let an immediately preceding echoed READ process to complete before typing. TYPE 2 is an entry which results in RF.1 being typed out in hex form as two hex digits. Each 4-bit half is converted to an ASCII hex digit (0-9, A-F) and separately typed out.

Notice that the READ routines are designed to facilitate repeated calls on READAH, while the TYPE routines are designed for repeated calls to TYPE5. In

Table I—UT62 Subroutine Calling Sequence

Entry Name	Absolute Address	Description
READ	813E	Input ASC11→RF.1,D (if non-standard linkage).
READAH	813B	Same as READ. If hex character, DIGIT→RD (see text).
TYPED	8198	1.5 bit delay. Then TYPE function.
TYPE5D	819C	1.5 bit delay. Then TYPE5 function.
TYPE5	81A0	Output ASCII character at M(R5). Then increment R5.
TYPE6	81A2	Output ASCII character at M(R6). Then increment R6.
TYPE	81A4	Output ASCII character at RF.1.
TYPE2	81AE	Output hex digit pair in RF.1.
TIMALC	80FE	Read input character and set up control byte in RE.1. Initialize RC to point to DELAY1.
DELAY1	80EF	Delay, as function of M(R3) (see text). Then R3 + 1.
OSTRNG	83F0	Output ASCII string at M(R6). Data byte 00 ends typeout.
MOVE	8429	See text.
FILL	824B	See text.
INIT1	83F3	See text.
INIT2	83F6	See text.
GOUT62	83F9	See text.
CKHEX	83FC	See text.
CALLR	8364	SCRT call routine.
RETR	8374	SCRT return routine.

Notes

(1) All routines, except DELAY, use R3 as program counter, exit with SEP5, and modify registers X, D, DF, RE, RF and location M(R2).

(2) DELAY routine uses RC as program counter, exits with SEP3 after incrementing R3, and alters registers X, D, DF, and RE.

(3) READ and READAH exit with R3 pointing back at READAH.

(4) All five TYPE routines exit with R3 pointing at TYPE5. All TYPE routines modify register RD.

(5) R0, R1, and R4. 1 are altered by UT62 while storing registers.

Table II—UT62 Register Utilization

Register Name	Register Number	Function and Comments
SP	R2	Stack pointer. UT62 sets R2 = 8CFF
PC	R3	Program counter for UT62.
CALL	R4	Call routine counter.
RETN	R5	Return routine counter.
LINK	R6	Subroutine data link.
CNT	RA	Byte count.
PTR	RB	Table pointer used by UT62
SRC	RB	Source register
DELAY	RC	Program counter for delay routines. Points to DELAY1 in memory.
ASL	RD	Hex input register.
DEST	RD	Destination register. RD.0 is used by all TYPE routines.
AUX	RE	RE.1 holds time constant and echo bit. RE.0 is used by all read and type routines and by TIMALC, OSTRNG, and CKHEX.
CHAR	RF	RF.1 holds input/output ASCII character. RF.0 is used by all read and type routines and by TIMALC, OSTRNG, and CKHEX.

order to output a string of variable data characters following a READ, given the timing restriction mentioned earlier, it is most logical to call TYPED or TYPE5D first, using an immediate "punctuation" byte (i.e., non-data such as space, null, etc.) to get the required initial delay and to follow either with repeated calls on TYPE (with the output variable data characters picked up from RF.1) or repeated calls on TYPE5 using immediate data characters. This procedure permits a maximum output character rate.

Another routine, **OSTRNG**, can be used to output a string of characters. OSTRNG picks up the character string pointed to by R6 and tests each character for zero. The characters should be already encoded in ASCII. If a zero is found (ASCII 'null'), the program terminates and returns control to the user via a SEP R5. If the character is not a zero, it is typed out to the terminal. The OSTRNG routine uses TYPED so that it may be called at any time - even following a read.

Tables I and II summarize the functions and calling sequences just described.

Examples of UT62 Usage

The following examples should help clarify how to use the UT62 read and type subroutines. Most examples use the standard subroutine linkage which requires that R2 point at a free RAM location.

READ Routines

This sample program will read ASCII-hex characters into register RD translating them from ASCII to hex in the process. Reading will terminate when a carriage return is entered. Entry of a non-hex digit other than a carriage return will cause a branch to an error program. This sample program as called by use of the standard subroutine call and return linkage. (# indicates hexadecimal notation)

```
READAH = #813B
LOOP: SEP R4; ,A (READAH) ..Call the hex
                          ..read program
      BDF LOOP   ..As long as ASCII hex
                 ..digits are entered
                 ..Read and shift in
                 ..Fall through if not hex
                 ..character
      GHI RF     ..See what character was last
                 ..entered
      XRI #0D    ..What is carriage return
      BNZ ERROR..If not, BR to error
                 ..Characters entered are now
                 ..in RD
```

The READ routine (at 813E) could be used similarly to enter characters; however, READ only enters them one at a time into RF.1 (and D) writing over the previous entry. Note that, even though incoming data is entered into D, the subroutine return program alters D. Therefore, valid data will only be found in RF.1 (and RD when READAH is used) if the standard subroutine call and return programs are used. An alternative technique is to use R5 as the main program counter (because all read and type routines terminate with a SEP R5) and call the program with a SEP R3 (because all read and type routines use R3 as their program counter). Example I below under TYPE Routines illustrates this technique.

TYPE Routines

Three examples using the various type routines follow.

Example 1: This program outputs a single character using the TYPE5 routine. It uses R5 as the program counter.

```
LDI#81   ..Set R3 to TYPE5 routine
PHI R3
LDI #A0
PLO R3
LDI #FF  ..Set R2 to free RAM location #3FFF
PLO R2
LDI #3 F
PHI R2
SEP R3   ..Call type
,T'R'    ..An "R" will be typed
yy       ..Next instruction
```

The TYPE5D routine is used in the same way.

Example 2: This program outputs a character using the TYPE6 routine. Note that R6 should be the program counter for the program calling TYPE6 if the character to be typed is an immediate byte because TYPE6 must always be from M(R6). But, because TYPE6 exits with SEP 5, TYPE6 must always be called using standard subroutine linkage for typing an immediate byte. An alternative is to use R5 as the main program counter but point R6 at the memory location containing the byte to be typed. This example uses standard subroutine linkage.

```
SEP R4   ..Branch to call routine
,#81A2   ..Address of TYPE6
,T'?'    ..Byte to be typed out
yy       ..next instruction
```

Example 3: The TYPE and TYPE2 routines pick up the byte in RF.1 for typing. TYPE simply outputs the character, whereas TYPE2 considers RF.1 a hex digit pair which it encodes in ASCII before typing. This example types out the hex digits 'D5', and uses standard subroutine linkage.

```
LDI #D5..Load hex digits D5
PHI RF  ..Into RF.1
SEP 4   ..Call TYPE2
,#81AE
yy      ..Next instruction
```

The TYPED routine is used the same way.

Note that all type routines, except TYPE2, expect the character they pick up to be already encoded in ASCII.

OSTRNG Routine

An entire message can be typed by using the OSTRNG routine. The ASCII bytes pointed to by R6 will be typed. When a '00' byte is detected, OSTRNG returns to the caller. This example will output the string

RCA COSMAC
MICROPROCESSOR

The standard call and return linkage is assumed.

```
OSTRNG = #83F0
.
.
.
SEP R4; ,A(OSTRNG)        ..Call OSTRNG
DC,T'RCA COSMAC'          ..1st Line
  ,#0D0A                  ..(CR)(LF)
  ,T'MICROPROCESSOR'      ..2nd Line
  ,#00                    ..End of Text
.
.
.
```

Memory MOVE Routine

This routine allows the user to move a block of data from one area of memory into another area. There is no restriction as to the direction of the move, and the areas may overlap. Before a call can be made to this subroutine, the lowest address of the source data must be placed in register RB, the lowest address of the destination in register RD, and the number of bytes to be moved in to register RA. MOVE returns to the user via SEP R5. The following example uses the Standard Call and Return Technique.

```
Move = #8429
.
.
.
LDI #02                   ..Source address
PHI RB
LDI #FO
PLO RB
LDI #03                   ..Destination address
PHI RD
LDI #F0
PLO RD                    ..No. of bytes
LDI #00                   ..To move
PHI RA
LDI #10
SEP R4; , A(MOVE)         ..Call MOVE
yy                        ..Next instruction
.
.
.
.
```

This sequence will move 16 bytes of data from #02F0 to #03F0.

Memory FILL Routine

An area of memory can be initialized by a known value by use of the FILL Routine. Registers RA, RB, and RD must be initialized before the call is made. RA must contain the number of locations to be filled, RB must contain the lowest address of the area to be filled, and RD.0 the data to be placed in memory. FILL also returns to the user program via SEP R5.

The following example, using SCRT, will initialize one page of memory to #00 beginning at address #0200.

```
FILL = #824B
.
.
.
.
LDI #01                   ..Byte count
PHI RA
LDI #02                   ..Destination
PHI RB                    ..Address
LDI #00
PLO RA                    ..Byte count
PLO RB                    ..Destination
PLO RD                    ..Data
SEP R4; ,A(FILL)          Call FILL
yy                        ..Next instruction
.
.
.
.
```

ASCII to Hex Conversion Routine (CKHEX)

The ASCII to hex conversion routine, **CKHEX,** examines the ASCII character in RF.1. If this character is not a hex digit, CKHEX returns to the user (via SEP R5) with DF = 0. If the character is hex, CKHEX returns with RE.0 = hex digit, DF = 1 and with the digit shifted into the least significant 4 bits of register RD. CKHEX uses the registers described above and, as with the other routines, is most readily handled via the standard call and return techniques. CKHEX is located at 83FC.

Initialization Routines

Two routines are provided, **INIT1** and **INIT2,** that initialize CPU registers for the standard call and return technique. These routines set up registers as follows:

R2 = R(X)	-pointing to 8C3F
R3	-will become the program counter on return
R4	-pointing to the call routine in UT62
R5	-pointing to the return routine in UT62
RC	-pointing to the delay routine in UT62

The only difference between INIT1 and INIT2 is the location to which they return. INIT1 returns to location 0005 with P = 3, while INIT2 simply returns by setting P = 3 and assumes that the user has already set R3 pointing to the correct return point. These programs are intended as a convenience to free the user from generating the overhead code required by the standard subroutine technique. They may also be used as an integral part of custom support programs running on the MCDS. Their absolute addresses are INIT1 = 83F3 and INIT2 = 83F6. Refer to Appendix A, the UT62 listing, for the absolute addresses of CALL and RET, which will be loaded into R4 and R5 respectively.

Following are examples of the use of these programs:

Example 1: Using INIT1
INIT1 = #83F3

```
Address Code Mnemonics Comment
0000    71   DIS,#00    ..Disable interrupts
0001    00
0002    CO   LBR INIT1  ..Initialize registers
0003    83
0004    F3
0005    --   USRPGM:    ..User program starts here;
                        ..p = 3
```

Example 2: Using INIT2
INIT2 = #83F6

```
Address Code Mnemonics Comment
0000    71   DIS,#00    ..Disable interrupts
0001    00
0002    F8   LDI A.1    ..Set R3 to return
             (START)
0003    00             ..point
0004    B3   PHI R3
0005    F8   LDI A.0
             (START)
0006    50
0007    A3   PLO R3
0008    C0   LBR INIT2  ..Branch to INIT2
0009    83
000A    F6
0050    --   START:--   ..User program starts here
                        ..P = 3
```

UT62 Reentry

UT62 can be reentered by means of a branch instruction to address 87F0H. There are no restrictions.

Tape READ and WRITE Subroutines

UT62 contains two subroutines, READ TAPE and WRITE TAPE, of interest to users who wish to write programs that communicate with the cassette tape hardware. These subroutines perform read and write operations byte by byte in a serial manner. The routines are evoked by the standard call and return technique described in the **User Manual for the CDP1802 COSMAC Microprocessor,** MPM-201.

READ TAPE

The READ TAPE subroutine at location 8502H is used to transfer bytes of data one byte at a time from the tape's buffer into register RF.1. A three-byte memory table, given in Table III, holds the current block number, the tape drive selected, and the current byte count in the selected block. At the time the call is placed, register C should be pointing at byte 3 of this table. On return, register RF.1 contains the byte read from tape and the table is updated by 1 byte. Register RC is unchanged. When the byte count reaches 384 (180H), it is reset to 0, the block number is incremented, and the next block is loaded from tape to the buffer.

To call the READ TAPE subroutine, the user's responsibility is only to create a 3-byte table in RAM, set the byte and block count to 0, and set the appropriate tape drive bit. As an alternative, the FNDTAP routine in UT62 can be used. The user should then point register RC at byte 3 of the table and call the subroutine. UT62 takes care of updating the table. If a parity error is encountered, UT62 issues the following message:

PARITY BAD THIS BLOCK

The user can either abort further calls at this point or continue. Also, DF will be set to 0 on return from the read that generated the parity message. Note that the tape should be rewound at the beginning of the first read and the play key depressed.

WRITE TAPE

The WRITE TAPE subroutine at location 8500H is used to write the byte in register RF.1 into the tape write buffer and then to tape. When the tape buffer is full (384 bytes), the write routine dumps this block out to tape. The requirements for WRITE TAPE are similar to those for READ TAPE. A 3-byte table in RAM selects the tape to be written to. The write subroutine takes care of updating the byte and block counts.

To call the WRITE TAPE subroutine, the user needs to create a 3-byte table in RAM as shown in Table III. The user should set the block and byte count to 0 and set the desired tape drive bit to 1. As an alternative, the FNDTAP routine in UT62 can be used. Then, put the byte to be written in register RF.1 and set register RC to point to table byte 3. A call to write (standard call and return) places the byte in the tape write buffer. On return, register RC is intact and the byte and block counts are updated. Because the WRITE TAPE subroutine stores blocks of data on

tape, the user needs to fill the block after writing the last byte to the buffer to assure that the last block is written to tape. The block size is 384 (180H) bytes. Successive calls to write for 383 bytes (block-1) would assure the last block transfer. The user can also examine the byte table to see how many bytes remain in the block and write out that quantity. Another technique is to change the table to a count of 383 (17FH). This technique will write the final byte in the block buffer and cause the buffer to be written to tape.

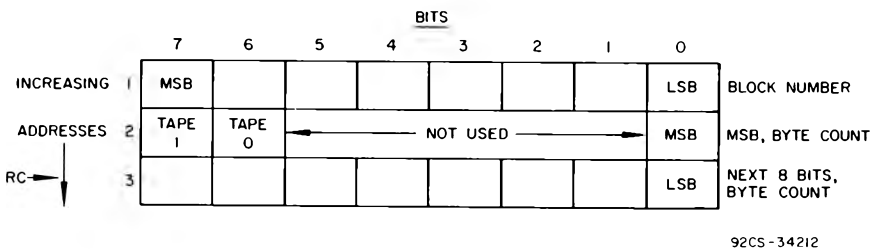
REWIND TAPE

The TAPE REWIND subroutine at location 8504H is used to enable the "REM" input on Cassette Tape units. The prompt

REWIND, THEN HIT ANY KEY

is written on the Video Monitor screen.

Table III—Read/write RAM table



3. Running BASIC3

BASIC3 is a full floating-point BASIC in ROM with cassette tape I/O capability. Entry into BASIC3 is accomplished while UT62 is running by the user typing B. After responding with the title, BASIC3 asks the user if the entry is "warm or cold." (C/W?) A warm start enables the user to re-enter BASIC3 with an existing program and data intact. A cold start clears all program and data storage and should be used whenever there is no existing program in memory.

While in BASIC3, the user can create and debug a BASIC program and save it on tape in a compressed form called pseudocode. Line editing is also available in BASIC3 to enable program modification without requiring the retyping of a new line.

On the CDP18S694, programs can be created, edited, and loaded by using the text editor. The editor stores text in ASCII code on tape which can then be loaded into BASIC3 by the user declaring the tape as the input device through the DIN command.

BASIC3 also allows calls to user machine language subroutines and provides for input/output instructions to any COSMAC Microboard device. For example, BASIC3 can talk to an external UART Module such as the CDP18S641 or input data from an A/D Converter such as the CDP18S642.

For further information on BASIC3, the user should refer to the manual **BASIC3 High-Level-Language Interpreter CDP18S841 User Manual,** MPM-841.

4. Running the MCDS COSMAC Resident Editor (CRE)

The text Editor supplied with the CDP18S694 and available as an option for use with the CDP18S693 allows the user to create ASCII files on cassette tape. These files can be Level I CDP1802 language, BASIC3 instructions with line numbers, or simply text. The Editor, in ROM, is evoked while UT62 is running by the user typing E. Upon this command, the Editor searches memory starting at 7FFFH for the top of free RAM. After finding the top, the Editor types its title and a prompt (→) thereby indicating its readiness to accept commands.

Introduction

After the user has written his COSMAC assembly language program and wants to assemble and run it, he immediately faces the problem of converting the hand-written source file into a machine-readable form. This conversion involves a keyboard-to-tape operation in which lines on the coding sheet are transcribed to become lines on a source tape. The Editor provides assurance that the created files are in proper format for later reading by the Assembler and for later modification, if necessary, by the Editor. Details on formats are given later in this Chapter.

Once a source file has been created and a first Assembly run made, it is very likely that error diagnostics will be returned by the MCDS COSMAC Resident Assembler (CRA) asking for corrections to the source file to conform to CRA's rules. Typically, the changes required at this point are "trivial" but necessary. For example, spaces may have to be removed in one or more expressions. The same symbol may have been erroneously used for two purposes. An operation mnemonic may have been misspelled or a punctuation character such as a comma, colon, or single quote omitted. The number of possible trivial errors is clearly large.

To correct the errors and to alter the source file to conform the program to the CRA rules, the Editor is used. Typically, modifications at this point merely involve insertion and deletion of single characters or replacement of a small string of characters by a substitute string. The erroneous source file is used as an input to the Editor and the user generates a corrected source file as an output. The new file is then re-assembled. At this point other trivial errors may appear which were not apparent to CRA on the first run. For example, an erroneous instruction operand may not have been flagged on the first assembly because its associated statement label or operation mnemonic may have also been in error. Thus, a new Edit-Reassemble pass may be necessary. Finally, a program is developed to which CRA does not object. At this point, a first run can take place.

The probability of a logical error in the program depends on its length and the previous experience of the programmer. Assuming one or more logical errors are found (via some "debugging" procedure), the source file must again be modified. Often such modifications are no longer trivial. For example, it may be necessary to find all instructions which branch to a given location and precede some of them with one or more instructions currently not in the program. Often, it may be necessary to delete some code or insert some code or move some code to a different point in the program. Several duplicated sets of in-line instructions may have to be removed and replaced with calls to one common subroutine which is to be added. The user may decide to "clean up" the program logically, in any one of several ways, or to improve its "readability" by modifying its comments or statements formats (by inserting TAB's or SPACE's, for example).

Such modifications to the source file also involve use of the Editor. After they are completed, a reassembly may again turn up new errors of the "trivial" variety. And so on. Thus the generation of a bug-free program typically involves the chart shown in Fig. 4. It is quite likely that the amount of time spent "conversing" with the Editor will be much larger than that spent with the Assembler.

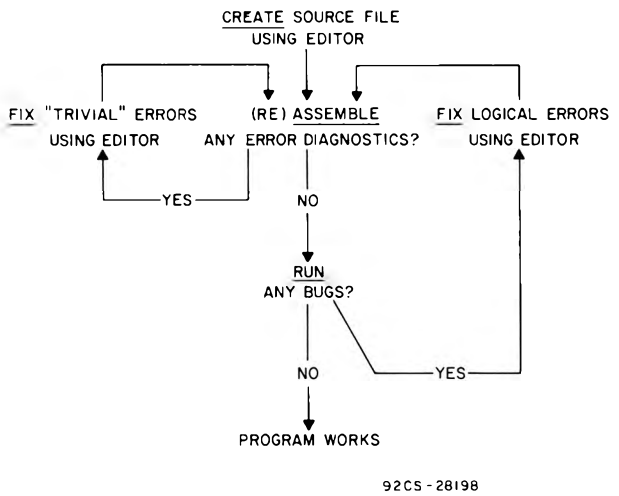


Fig. 4—Flow chart for "bug-free" program.

A source program may be viewed as a long sequence of characters. When the COSMAC Resident Editor reads the source file, it places this character sequence in memory, with the code in each memory byte representing one source program character. The user is then free to type commands to the Editor to manipulate the memory representation of the program. For example, the user may identify a specific location and specify a character sequence to be inserted there. He may also identify certain characters to be deleted or altered. He may ask the Editor to search for the occurrence of specific character sequences, after which further memory modifications (corrections) may be made. (Details of available commands are given later).

After he is satisfied that the new memory representation of the file contains all the desired changes (frequently the user begins an editing session with a hand-written list of the changes to be made), he asks the Editor to write (create) a new file containing the new version of the program. This new file is then used as the input file for a reassembly.

CRE Operating Considerations

Memory Space Requirements

CRE requires about 100 bytes of the RAM work space for its own internal purposes. The remainder of the available RAM space is used as an editing area called a buffer. Virtually all CRE operations involve the buffer. CRE is designed to take advantage of all of the available RAM space for its buffer area.

All but the 100 bytes of work space are available for this purpose in the 4 kilobytes of RAM supplied. If more RAM is present in the system, CRE will automatically add it to its buffer area. It tests for additional RAM the same way CRA does.

Input and Output Files

Normally, a user creates a file using CRE by filling the buffer from the I/O terminal keyboard and then causing CRE to write this information onto an output tape (which will contain the created file).

An existing (input) file may be modified (edited) by reading portions of it into the buffer, then using CRE commands to alter the contents of the buffer, and finally writing the results onto the output file. Typically, the output file is a new version of the input file. After an editing session, the new version is retained and the old version is discarded (although it may be temporarily saved for future reference or backup).

Thus CRE has means to read an input file into the buffer, means to examine and modify the contents of the buffer in many ways, and means to write the buffer contents onto an output file. Alternatively, when an input file does not exist, the user creates an output file by loading the buffer from the keyboard.

Record Formats

In order to understand the various commands which CRE is designed to execute, it is fundamentally important that the user understand how information is normally recorded on the I/O media (tapes) and in the buffer.

A file is a sequence of records or lines. Each line consists of a sequence of characters. The length of a line is restricted to 78 or fewer characters of data. Thus, a line in a file is normally printable as a line on the I/O terminal printer. Each character is represented by an 8-bit code or byte, either on the tape or in memory. Typically, every character in a line is a printable character (including space or blank). Every non-printing character code represents a control character. A control code may be generated on the keyboard either by hitting an appropriately marked key (e.g., RETURN, ESC, etc.) or by depression of the CTRL button while hitting another key. The printer reacts to the receipt of a control character in one of several possible ways. Some control characters (such as carriage return, line feed, bell, etc.) cause the printer to execute a specific control function. Other control codes either are ignored by the printer or may cause the equivalent of a space on the printed line.

A line in a file may contain control characters (with certain restrictions to be discussed later). If it does, it is quite possible that its printed record will not completely reflect the true contents of the line. CRE treats most control characters which it encounters within a line in the same manner as it treats printing characters. However, certain control characters have special meaning to CRE.

The proper format for tape files is shown in Fig. 5. Each line is terminated with a CARRIAGE RETURN (CR), LINE FEED (LF) pair, followed by a field of six nulls. The NULL character (hex code 00) is ignored by the system. A set of nulls appears after each CR, LF, pair merely to provide a sufficient time delay for the printer carriage to settle to the new line when the tape contents are being printed. Note that the last line on the tape should be followed by a "dummy" line containing only the single data character DC3. DC3 is a special control character (generated on the keyboard by hitting CTRL and S). It acts as an END OF FILE indicator.

Tape records read by CRE are deposited into the buffer as they appear on the tape, but with all LF's and NULL's ignored. (Incidentally, the DEL or RUBOUT character, hex code 7F, is also ignored on tape input) While CRE operates on the data in its buffer, it specifically uses the CR character as an indicator of the end of a line. (Recall that a line has a variable length.) A new line is assumed to start with the next character in the buffer. The buffer format is shown in Fig. 6.

When CRE is depositing keyboard data into its buffer, the ASCII code equivalent of each struck key (any printing character, and almost any control character - with exceptions as noted below) goes into memory and is also "echoed" back to the printer. However, CRE specifically ignores the LF key. Further, when the RETURN key is hit, the CR character goes into memory and a CR, LF pair of characters is echoed back to the printer to start a new line. Thus, the user terminates a line of keyboard input with a single carriage RETURN. Normally, then, the LF character should not appear at any point in the buffer.

Whenever CRE transmits a CR character to the terminal, it automatically appends to it the LF, six NULL field to maintain the tape format just discussed.

It is conceivable that due to the user error one or more lines on the input file or in the buffer may exceed the 78 data character length restriction. For example, data alterations in the buffer may have resulted in deleted CR's. (Note that each CR deleted in the buffer causes the concatenation of its adjacent lines.) CRE has the following provisions for handling lines that exceed the length restriction:

(1) Whenever CRE is outputting a line to the printer as the result of a user TYPE command, if the line exceeds 78 characters, a "LINE TOO LONG" message will also be printed.
(2) IF CRE encounters too long a line while writing from the buffer to the output tape, the line will be broken up, using as many 78-data-character records as are necessary - each terminated by an appended CR, LF, 6-NULL field.
(3) A line which is too long on the input file is truncated to 78 characters, with a CR appended, in the buffer.

Buffer Pointer

The total RAM space available for the buffer is generally partially filled. When CRE is first initialized, the buffer is empty. When data is added to the buffer (from the keyboard or from the input tape) the buffer expands. When data is deleted, the buffer contracts. CRE continually keeps track of the present extent of the buffer within the work space.

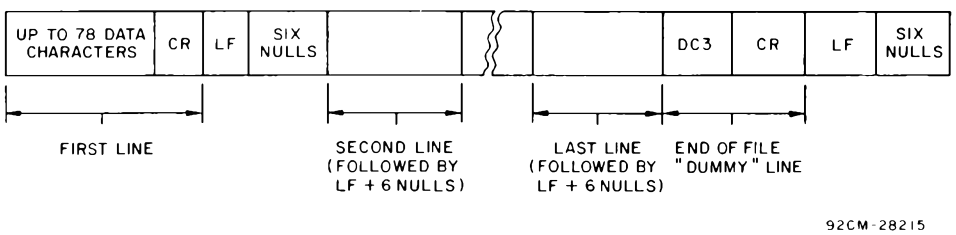


Fig. 5—Tape file format.

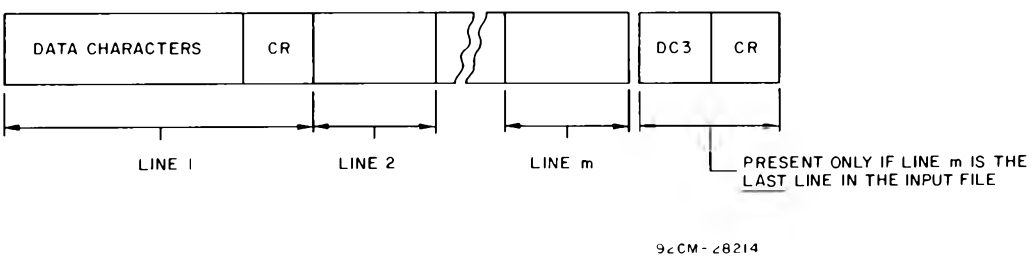


Fig. 6—Memory buffer format.

CRE maintains a virtual pointer which identifies some point between two characters in the buffer. This pointer has the same function as what is commonly called a "cursor". Most CRE operations are executed relative to this pointer. Further, several CRE operations exist specifically to alter the location of the pointer. It is essential, therefore, that the user be aware at all times of the current location of the pointer. Because the pointer is not visible, it is the user's responsibility to keep track of where the pointer is. Often, its location is verified by asking CRE to type information in the buffer at the current pointer position. Alternatively, the user may first initialize the pointer to a known reference point (e.g., the beginning or end of a line, or the beginning or end of the buffer) and then move it relative to this known origin.

In illustrative examples, the location of the pointer is indicated with an arrow below and between the two buffer characters. For example, in

ABCDE
↑

the character before the pointer is B and that after the pointer is C.

Unless otherwise noted, whenever text is deleted from the buffer, the character sequence to be deleted exists either immediately to the right or immediately to the left of the pointer. After the deletion, the buffer has contracted by the number of characters deleted. If the field deleted is to the right of the pointer, the character immediately to the left of the pointer remains the same. The character to the right of the pointer then becomes the character that was immediately to the right of the deleted field. A corresponding statement can be made for deletion to the left of the pointer.

When text is inserted, the buffer expands. Unless otherwise noted, text is inserted between the two characters at the position of the pointer. After the insertion, the pointer is positioned immediately after the inserted text. Thus, the character to the right of the pointer remains the same.

The execution of many CRE operations starts at the present pointer position and proceeds either towards the end or towards the beginning of the buffer. CRE insures that the pointer cannot be moved past the present limits of the buffer. If the pointer reaches the beginning or the end of the buffer, the operation stops - leaving the pointer at that point. For example, if the pointer is positioned n characters from the end of the buffer and the user asks to move the pointer m characters to the right, with m>n, then the operation will stop after the buffer pointer has been incremented by only n.

CRE Command Operation

Command Strings

On entry, CRE types the following message:

MCDS TAPE EDITOR VERSION X.X

and then follows with its user prompt —>. The —> prompt always indicates that CRE is ready to receive a new user command from the keyboard and has executed the previous one.

After receiving the —>, the user types a sequence of one or more commands which CRE will execute in order. Most commands may be optionally delimited (ended) by a special termination control character. Commands which include text arguments of variable length must include this character to define the end of a text field. A CRE command string is always terminated by the occurrence of two successive ESCape characters or its equivalent (the control character whose ASCII code is a hex 1B.

The I/O terminals discussed here, operate in the full duplex mode, in which the data path from the keyboard to the MCDS is distinct from that between the MCDS and the printer. Normally, a program merely "echoes" back to the printer what it has just received from the keyboard. However, whenever CRE receives an ESC character, it is echoed back to the printer followed by a $ - giving a visual indication of the ESC key depression. Thus, a typical command string normally appears on the printer as

COMMAND1$COMMAND2$ ... COMMANDn$$

where in most cases the separating ESC's are optional but the final pair is mandatory. (If ESC is not hit, the $ will not appear, of course.) A command string must be terminated by two depressions of the ESC key.

Command Formats

The heart of the command is a single letter mnemonic (such as "T" for TYPE, "I" for INSERT, etc.). In many cases, this letter may be optionally preceded by a decimal number (later denoted by n) indicating the number of characters or lines involved. Further, in some cases this number may be preceded by a minus sign (-) indicating a direction (from the present pointer position) toward the beginning of the buffer rather than toward the end (as is normally assumed). If no number is present, CRE assumes the value 1.

Given an arbitrary pointer location, the possible CRE interpretations for n are normally as follows:

(1) **Character Operations:** Positive n identifies the n characters to the right of the pointer (including control characters). Negative n identified the n characters to its left. Unless otherwise noted, n=0 results in no operation.

(2) **Line Operations:** Positive n identifies all characters to the right of the pointer up to and including the nth CR encountered. If the pointer is in the middle of a line, the first line will constitute only the remainder of that line. Negative n identifies all characters to the left

of the pointer up to but not including the -n+1st CR. If the pointer is in the middle of a line, the last line (in this set of lines) will consist of only those characters in the present line to the left of the pointer. Thus, n=0 specifically indicates the portion of the present line to the left of the pointer.

In certain cases a command mnemonic letter is followed by one or two variable-length text arguments (whenever the user needs to specify some sequence of characters to insert or to search for). All such arguments must be terminated by the ESC character (echoed as $). In subsequent discussion, an arbitrary text argument will be denoted by a symbolic statement such as "*text*".

Correct Command Typing Errors

A typing error in a command string may be corrected by use of the RUBOUT (DEL) character to "erase" previous characters already typed. Do not use backspace. Each time CRE receives a RUBOUT within a command string, it erases the last character from its stored version of the command string. Further, it echoes back to the terminal the character just erased. For example, suppose the user types the command string ABC$DE (each of the letters is a valid command mnemonic) followed by four rubouts. On the terminal, he would see

ABC$DEED$C

where the last four characters were those erased. The characters AB would then remain in CRE's stored command string register.

Clearly, any such erasures must occur before the double ESC character, which terminates the command string, is struck.

If CRE finds an invalid command while in execution of a command string (i.e., after the user has typed the double ESC), it returns to the user the error message

BAD COMMAND??"xxxx..xx$"

where xxx..xx reproduces that part of the command string which has not been executed.

Interrupting CRE Execution

The user may at any time stop CRE execution by depressing and holding the BREAK key on the keyboard. This key is used, for example, to stop a long typeout. On receipt of the BREAK, CRE stops execution at whatever point was reached and returns to the command input mode by issuing another prompt. To assure the clean entry of succeeding commands, the **RUBOUT** key should be depressed to erase any erroneous noise characters that may have been entered as a result of the break.

After a BREAK, the user should normally verify or reinitialize the buffer pointer position before resuming further editing.

Filled Work Space Warning

If CRE determines that a command string threatens to use up the remaining work space, it will stop echoing keyboard input characters to the printer and will echo instead the BELL control character causing the I/O data terminal to ring its bell as a warning. The user should respond by erasing part of it with the RUBOUT key.

If CRE runs out of space during command execution, it will return the error message

MEMORY FULL "xxx...xx$"

where, again, xx...xx is a reproduction of the unprocessed part of the command string.

CRE Commands

This section contains a summary of the individual commands which CRE is designed to recognize. Each command is described with a specification of its acceptable format and an explanation of its execution.

Single Commands

Pointer Control Commands

BEGINNING
Format: B
Execution: Pointer repositioned to the beginning of the buffer.

END OF BUFFER
Format: Z
Execution: Pointer repositioned to the end of the buffer.

CHARACTER STEP
Format: nC
Execution: Step pointer right (or left) by n characters.†

LINE STEP
Format: nL
Execution: Step pointer down (or up) by n lines

Reading the Input Tape

DEFINE READ TAPE
Format: R
Execution: After the command, CRE asks: READ? The user enters 0 or 1, signifying appropriate tape, followed by (CR).

†A positive (unsigned) n indicates the direction of right or down; a negative n indicates left or up for all commands.

APPEND
Format: A
Execution: Lines are read from the input file (continuing from the last line) and appended to the end of the buffer. The operation continues until one of the following occurs:

(1) End of file character detected (i.e., last line has been read).
(2) 3/4 of the remaining available space has been filled.
(3) 50 lines have been transferred.

The pointer is repositioned to the beginning of the first appended line. In large memory systems, multiple appends may be used to bring additional lines into the buffer.

Note: The keyboard BREAK key is ignored during execution of this command only.

Deletion Commands

DELETE
Format: nD
Execution: n characters right (or left) adjacent to the pointer are deleted.

KILL
Format: nK
Execution: n lines right (or left) adjacent to the pointer are deleted.

Text Insertion and Data Manipulation

INSERT
Format: I*text*$
Execution: Typed *text* is inserted to left of present pointer position. The *text* may contain multiple lines.

SAVE
Format: nX
Execution: Copy n lines adjacent to the pointer into a special SAVE area external to the buffer. The pointer position is not changed. Previous contents of the SAVE area are overwritten. CRE types CAN'T SAVE if there is insufficient room in the SAVE area and it does not save any lines. CRE clears the SAVE area if n=0 (zero).

GET
Format: G
Execution: Equivalent to an INSERT, but using the present contents of the SAVE area as an implicit text argument. Note: SAVE and GET are especially useful in sequence as a copying mechanism - to MOVE text.

CRE dynamically allocates the available RAM work space to its SAVE area, stack area, and the buffer or editing area. Once lines have been SAVE'd, they remain in the SAVE area indefinitely until the next SAVE command overwrites them. If many characters have been SAVE'd, the area available for the buffer will be proportionally reduced. The SAVE area is not automatically cleared by a GET command. Several GET commands may be issued against the same SAVE area. It is good practice, therefore, to clear the SAVE area when it is no longer needed in order to make that area available to the buffer. This step is accomplished by typing 0X (zero-X).

If an attempt is made to save more lines than there is room for. CRE will type

CAN'T SAVE

and will not transfer any lines to the SAVE area.

FIND
Format: F*text*$
Execution: A search for the specified character sequence *text* occurs from the current pointer position toward the end of the buffer. It stops either when a match is first encountered or when the end of the buffer is reached. In the first case, the pointer ends positioned immediately after the matching string. In the latter case, a "CAN'T FIND" message is printed, and the pointer position is unchanged.

SUBSTITUTE
Format: S *search text* $ *substitute text* $
Execution: Operates as FIND does above (using *search text* as the search argument). However, on a match, the *substitute text* replaces the matching sequence - with the pointer positioned after the inserted text. The *substitute text* must not be omitted from the command.

Output Commands

DEFINE OUTPUT TAPE
Format: O (letter, not zero)
Execution: After the command, CRE asks: WRITE? The user enters 0 or 1, signifying appropriate tape, followed by (CR).

TYPE
Format: nT
Execution: Type the n lines adjacent to the current pointer. The pointer position remains unchanged.

WRITE and DELETE
Format: nW
Execution: n is treated as positive. The n lines at the beginning of the buffer are written to the output tape and also deleted from the buffer. The pointer ends up positioned at the beginning of the lines remaining in the buffer.

FILE CLOSE
Format: Y
Execution: Places an end-of-file marker (DC3) at the end of the working buffer, outputs the buffer to the write tape, and restarts CRE. This command is normally used to end a file-creation edit session. It should never be used for an edit of an existing file.

END

Format: E
Execution: The entire buffer is written to the output tape. Any lines remaining on the input tape are then copied to the output tape. CRE then reinitializes for a new editing session with buffer cleared and with the pointer positioned at the beginning of the work space.

PRINT ON LINE PRINTER

Format: nP
Execution: Print the n lines adjacent to the current pointer on a line printer. A CDP18S646 Microboard Parallel I/O Module and a suitable printer are required. The pointer position remains unchanged.

RETURN TO MONITOR PROGRAM

Format: U
Execution: The system returns to UT62 control.

QUIT THIS SESSION

Format: Q
Execution: The system restarts CRE.

Summary of CRE Commands and Controls Characters

A summary listing of the foregoing commands together with the meaning of each one is given in Table IV. A summary of the special CRE control characters is given in Table V. The CRE error messages are summarized in Table VI.

Composite Commands and Nesting

CRE also permits the user to specify composite commands. A composite command is a command string (one or more commands) enclosed within angle brackets (<...>). A command string may be preceded by a decimal number indicating the number of times that the string within the brackets should be executed.

Table IV—CRE Command Summary

Format	Meaning
B	Move pointer to BEGINNING of buffer.
Z	Move pointer to END of buffer.
nC	Step pointer right (or left) by n CHARACTERS
nL	Step pointer down (or up) by n LINES
A	APPEND lines to end of buffer from input tape. Reposition pointer to beginning of APPENDed area.
nD	DELETE n characters after (or before) pointer.
nK	KILL n lines after (or before) pointer.
I*text*$	INSERT *text* at present pointer position. (Position pointer after it.)
nX	Save n lines after (or before) pointer. (Pointer position unchanged.) Clears the SAVE area if n = 0.
G	GET the last SAVEd lines and INSERT them.
F*text*$	FIND the first occurrence of *text*, searching from present pointer position toward end of buffer. If found, position pointer after the match. If not, type CAN'T FIND.
S *search text* $ *substitute text* $	FIND *search text* and SUBSTITUTE *substitute text* for it.
nT	TYPE n lines after (or before) pointer. (No change in pointer location.)
nW	WRITE (and delete from buffer) the first n buffer lines on the output tape. (Pointer ends up at beginning of remaining buffer.)
Y	Used to end a file-creation session. Places an end-of-file marker at the end of the buffer then writes the buffer to tape.
E	END the editing session. Equivalent to an nW, with n equal or greater than the number of buffer lines, followed by a copy of remaining input tape to output tape.
R	Define input or read tape.
O	Define output or write tape.
P	Print n lines after (or before) pointer. (Pointer position unchanged
U	Return to UT62
Q	Quit and restart editor.

Table V—Summary of Special CRE Control Characters

(1) ESCAPE	Echoed as $. Optional command separator. Required after a TEXT field. Two required at the end of a command string.
(2) LINE FEED	Ignored on input. Inserted after CR on output
(3) CARRIAGE RETURN	Line terminator character. Stored in buffer.
(4) NULL	Ignored on input. Set of six inserted after LF on tape output.
(5) RUBOUT or DELETE	Erases previous character in a command string. Ignored on tape input.
(6) DC3	End-of-file character. Inserted by user at end of a created file or read in from an existing input file.
(7) HORIZ TAB	Echoed as 1 to 8 spaces when typed. Converted to 1 to 8 spaces on tape output. Can begin a command implying a previous INSERT.
(8) BREAK	Pressing BREAK will terminate a long operation. Next, press RUBOUT or DELETE to get a prompt→

Note:
Within a command string, but not within a text field, CRE ignores any inserted spaces or CR's. Spaces or CR's may be used to improve the readability of the command string, if desired.

Table VI—CRE Error Messages

Message	Meaning
LINE TOO LONG	A line that CRE is attempting to TYPE has more than 78 characters.
BAD COMMAND?? "XXX . . . X$"	CRE has found an invalid command in a command string. XXX . . . X is that part of the string not executed.
<BELL>	Filled work space warning.
MEMORY FULL "XXX . . . X$"	CRE ran out of work space during an execution. XXX . . X is the unprocessed part of the command string.
CAN'T SAVE	There is not enough room in the SAVE area.
CAN'T FIND	The specific character sequence was not found between the pointer's previous position and the end of the buffer.
ITERATION STACK FAULT	CRE ran out of stack during execution of a command string. May indicate improperly paired brackets in the string.

One composite command may include another. Thus, CRE permits the "nesting" of commands. For example,

B5<3C4<DI $>L>$$

causes replacement of the 4th through the 7th characters in the first 5 lines in the buffer by spaces. The pointer ends positioned at the beginning of the sixth line.

With nested commands, the user must be aware of the order in which commands will be executed and the number of times individual operations will occur. The following example should indicate the general algorithm. Other examples will be given later. Consider the command string

$$a<b<CS_1>c<d<e<CS_2<CS_3>CS_4>>$$

where the lower case letters represent numbers and where each CS_i represents an elementary command string (appropriate ESC characters must be inserted). Fig. 7 indicates CRE's flow chart for the execution of this command string. It is derived by properly pairing the angle brackets in the string. Notice, for example, that CS_2 is executed a number of times equal to the product of a, c, d, and e.

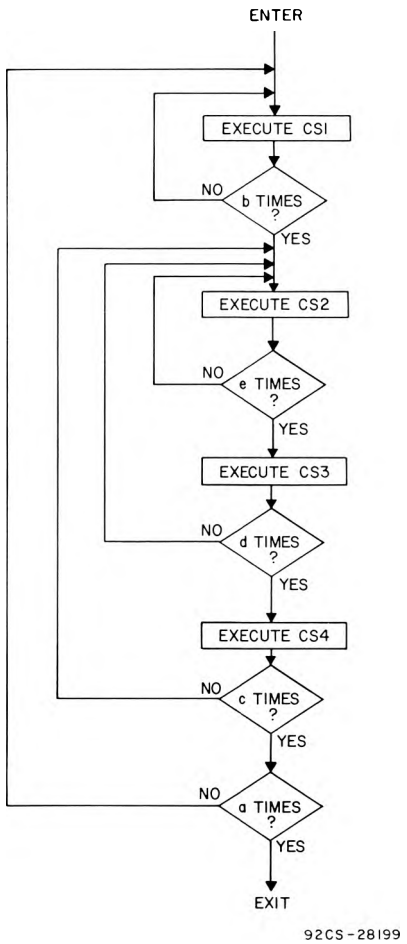


Fig. 7—Execution of nested composite command.

To execute a nested command, CRE maintains a stack in part of the available work space. The amount of stack space required depends on the depth of nesting in the command, i.e., on the number of loops within loops, as in Fig. 7, which in turn depends on the depth of bracket-pairs-within-bracket-pairs in the command string. If CRE runs out of stack space during execution, it will issue the error message:

ITERATION STACK FAULT

This error message is most likely to occur if the brackets in the command string are not paired properly. In particular, it occurs if a bracket is missing.

Note that if the user fails to terminate a *text* string with the required ESC character, all subsequent characters until an ESC does occur will be treated as part of the presumed *text* string. Thus, it is quite possible that a missing ESC in a nested command string could also result in the "improperly-paired-brackets" errors message, ITERATION STACK FAULT.

Horizontal Tabs

CRE assumes an implicit horizontal tab stop after every eight character positions in a line. If the user types a HORIZ TAB character (CONTROL and I) as part of a TEXT field, CRE will insert this character into its buffer, but it will echo back to the printer a sufficient number of spaces to reach the next implied tab stop. HORIZ TAB characters read from the input file are loaded into the buffer as is. On output, each HORIZ TAB buffer character is converted into the required number of spaces, extending the line length in the process. Thus, HORIZ TAB characters cannot appear on the output tape. The TAB character can be used to produce straight columns in a source file.

Note: As a special case, CRE interprets a text beginning with a HORIZ TAB character as if an INSERT command had preceded it.

Additional Note

Normally, the INSERT of a non-existent text field (i.e., the command I$) results in no operation. Further, it is normally illegal to precede an INSERT command with a numeric argument. However, the specific command nI$ (combining the two) is legal. It causes the insertion of a single character whose ASCII decimal value is n (module 128).

For example, 10I$ will cause insertion of a LF character (hex 0A).

Using CRE

In this section, information is given on the development and manipulation of a file through use of the Editor (CRE). In addition, some useful common sequences are given to illustrate CRE's data manipulation facilities.

Creating a File

A file is created by a repeated sequence of the following steps:

(1)File buffer from keyboard with sequence of INSERT's.
(2)WRITE buffer to output file.

A single I command may take as an argument a text string of arbitrary length. Thus, many lines may be inserted with a single I command. Each line is terminated by pressing the CR key. A typical INSERT will thus appear on the printer as

—> I *line 1*	..Insert command and 1st line of ..text.
line 2	..Additional lines
.	..Additional lines
.	..Additional lines

```
                ..Additional lines
line n          ..Last line of text
$$              ..End of insert; double
                ..ESCAPE
```

because each CR is echoed as CR, LF. Such commands may be sequenced until the buffer is nearly filled. These sequences are then normally followed by an nW (WRITE) command or BnP, with n $\geqslant$ the number of lines in the buffer. By use of the W command, the buffer is cleared after the WRITE to the output file and is ready for a new set of INSERT's.

The last line inserted should be followed by the insertion of a terminating dummy line consisting of the single character DC3 (CONTROL and S), followed by the usual RETURN to assure proper tape format, as discussed earlier.

Adding to a File

A section is added to an existing file by first copying the portion before the insert, and finally copying the portion after the insert. The first copy involves one or more APPEND's followed by WRITE's, up to the APPEND that reads in the section of the input file containing the insertion point. Note that appending to the end of a file may also be considered as an insertion just before the last DC3 terminating line.

Assuming the insert point is arbitrarily located within the buffer, several variations exist for adding text material. For any of these variations, the pointer must first be moved to the insert point. Then, a sequence of INSERT's is made at that point, particularly if the amount of the inserted material is small. Alternatively, one could SAVE all lines following the pointer (with an nX, n sufficiently large), delete them with an nK command, and then WRITE the data remaining in the buffer with an nW (n sufficiently large). The buffer then become empty with all records preceding the addition written to the output tape. Additional INSERT's and WRITE's may now be made. Finally, a GET followed by a WRITE will attach the material after the insert point. Now if there is more unread material on the input file, the GET may be followed directly by an END command. This command will automatically copy the remaining input tape.

In summary, one inserts material into an existing file by beginning with a copy sequence (a series of APPEND's followed by WRITE's). Then with the pointer positioned properly, one may execute nX nK nW (n sufficiently large). Now, one operates in the CREATE mode with INSERT's followed by WRITE's. Finally a GET or GnW will complete the sequence.

When appending to the end of a file, one has the alternative of removing, after the last APPEND, the dummy termination line via a Z-1K command string. Operation then is as in the CREATE mode. For this case, the final DC3 line must later be inserted to the end of the file.

Deleting a Section in a File

To delete a section in a file, first copy up to the deletion point, as previously discussed. Lines to be omitted may then be explicitly deleted from the buffer (by nK, with pointer properly positioned). If further lines to be deleted exist on the input file, further APPEND's are required.

Moving a Section in a File

Assume that the file section to be moved is sufficiently small. If the movement is toward the end of the file, the following sequence may be used:

(1) Copy input file up the section to be moved.
(2) SAVE the section to be moved. Then DELETE it in the buffer.
(3) Continue copying the input file up to the insertion point.
(4) GET and WRITE the SAVE'd section.
(5) Copy the remaining part of the input file.

If the movement is toward the beginning of the file, one must first find the section to be saved, SAVE it, DELETE it, and then reinitialize the input file. Then, the sequence of steps 3, 4, and 5 above will effect the insertion.

Several complications of this simple procedure can occur. First, the material to be moved may overlap two APPEND's. In this case, one does not SAVE until the second APPEND has been executed. Second, the material to be moved may consist of a substantial portion of the input file so large that it must first be copied on to a third temporary tape which might be called an "insertion file". If this condition exists, the user should be sufficiently familiar with CRE so that he will be able to create and use this special temporary file.

Modifying a Section in a File

By now the reader should be reasonably familiar with the commands APPEND, WRITE, END, INSERT, SAVE, and GET.

The most common use of CRE is to modify the contents of a file at a given point (typically, to correct an error). To make such a modification, the user must first read that section of the file into the buffer. Normally, a copy of the initial portion of the file is necessary, up to the APPEND which brings into the buffer the section to be modified. Now, the remaining CRE commands are available to effect the modification. After the change is made, the process is terminated with an END command or a WRITE command as appropriate.

Some Command Examples

Below are several examples of useful command

sequences to further acquaint the reader with CRE's data manipulation facilities. In each example a command string is given and followed by a short explanation of what it will do. For clarity all zeros are slashed (Ø) to distinguish them from the letter O in these examples.

(1) Assume the pointer is arbitrarily positioned within a line in the buffer:

ØLT Types the entire line leaving the pointer at its beginning.

ØTT Also types the entire line, but leaves the pointer unchanged.

ØK Erases the portion of the line to the left of the pointer.

K Erases the portion of the line to the right of the pointer.

ØLK Erases the entire line.

For each of the following command sequences, it is assumed that n is sufficiently large.

BnK Erases the entire buffer.

ØX Erases the entire SAVE area.

BnT Prints the entire buffer.

(2) Assume the pointer is positioned at the beginning of a line in the buffer,

nXnKZ-mLG

will move the next n lines to m lines from the end of the buffer and erase them from their original position.

(3) The command

Bn<mCI $L>

for n sufficiently large, inserts a field of spaces in all lines, at a point m characters from the beginning of each line.

(4) One can also scan the entire buffer with a FIND or SUBSTITUTE command by similarly using a sufficiently large numeric argument (called n below). The command will terminate when the end of the buffer is found with a CAN'T FIND message. For example:

Bn<S*field1*$*field2*$>

will replace all occurrences of *field 1* by *field 2.*

Bn<F*text*$-mD>

will delete all occurrences of *text*, if m= the length of the *text* field.

Bn<F*text*$0TT>

will print all lines containing *text.*

Bn<F*text*$0LK>

will delete all lines containing *text.*

Bn<F;ICR>

will break all lines containing semicolons into as many lines as there are semicolons—each terminating in a semicolon. (Note: In this case, any line originally ending in a semicolon will be followed by a "line" containing zero characters).

Bn<S $CONTROL I$L>

will replace the first space in every line in the buffer by a horizontal tab control character.

Bn<A50T50K>

will perform the following n times: append in the next (first) section, type it, and delete it from the buffer. This command string can be used to type a long file that can't be held all at once in the buffer.

5. Running the MCDS COSMAC Resident Assembler (CRA)

The MCDS Level I Assembler (CRA) supplied with the CDP 18S694 and available as an option for use with the CDP18S693 allows the user to convert a Level I source file on tape (source code) into an executable machine language program on another tape (object code). The object tape can then be loaded into memory by the UT62 monitor program (TAPE READ) for CPU execution, or it can be placed in an EPROM by the PROM programmer supplied with the CDP18S694 but also available as an option for the CDP18S693.

The use of the Assembler permits the programmer to write programs using convenient symbols and expressions. The input or source file consists of a sequence of statements. The Assembler converts these statements into hexadecimal instructions and data and writes them to an object file tape. Some statements are special commands to the assembler only. They produce no output code and are called directives.

An assembly language program (Level I) is by far easier to create and modify than a machine language program. Labels and comments in the source code help to produce more readable programs and aid in documentation. The Assembler also flags certain programming errors and thus creates machine code that is simpler to debug.

The following material describes how to use the Assembler and explains in detail the Assembler's treatment of a source file and the form of the Level I assembly language.

Using the Assembler

The Assembler is called from the UT62 monitor program by the user typing an A. The system searches memory for the top of free RAM, types the Assembler title, and then requests source (READ) and object (WRITE) tapes.

The Assembler requires two active tape drives to perform a successful assembly. While the UT62 is running, it is desirable, although not essential, for the user to rewind both tapes, source and object, before entry into the Assembler.

A sample assembly session is illustrated in Fig. 8 and should be referred to during the following discussion. At line 1 the user types an A to evoke the Assembler. The system responds at line 2 by typing the Assembler

```
1  *A
2  MCDS ASSEMBLER VER 0.0
3  READ? 0(CR)  [1(CR)]
4  WRITE? 1(CR)  [0(CR)]
5  PRESS PLAY ON READ TAPE
6  TYPE ANY KEY K  [ANY KEY]
7  H, L, U, J? H  [L]
8  REWIND, THEN TYPE ANY KEY K [ANY KEY].
9  PRESS PLAY ON READ TAPE  ..PRESS PLAY BUTTON ON TAPE 0
10 RECORD ON WRITE TAPE  ..PRESS RECORD BUTTON ON TAPE 1
11 TYPE ANY KEY TO START PASS2 K  [ANY KEY].
12 DONE
13 H, L, U, J? J
14 REWIND, THEN TYPE ANY KEY K [ANY KEY].
15 H, L, U, J? U
16 *
```

Fig. 8—Sample assembly session. System response is underlined. Sample user response is not underlined; alternate user response is in square brackets [ ].

title and at line 3 asks for the source tape drive number. When a carriage return (CR) is struck, the system checks the entry for a 0 or a 1. Either response sets the appropriate tape source. If an unacceptable number is the entry before the (CR), the system again asks for a 0 or a 1. Non-numeric characters are ignored by the Assembler.

At line 4 the system asks for the object tape. The same conditions apply here as with line 3 and the (CR) key sets the write (object) tape. In line 5 and line 6, the system tells the user to enable the source tape by pressing the play button on the cassette recorder and then to type any key to continue to line 7.

At line 7 the system asks for H (generate a hexadecimal file), L (generate a listing file), U (abort and go to UT62), or J (enable tapes for rewind). If H or L is typed by the user, the system starts PASS1 of the source tape to create a symbol table. After PASS1 is done, the system tells the user to rewind the source tape (line 8) and then type a key.

At line 9, the system tells the user to enable the read tape for PASS2 by pressing the play button. Line 10 alerts the user to press the record button for the object (write) tape. At this point, the typing of any key starts the system running PASS2. During the pass object code is generated (hexadecimal or listing) and written to the object tape.

Line 12 tells the user that the assembly is done and line 13 requests a new command. A J response enables both source and object tapes for rewind. A U response causes the system to exit the assembler and enter monitor UT62.

Location Counter

The basic function of CRA is to fill a simulated COSMAC memory (the object code area or file) with the hexadecimal equivalent of the user's source program. For this purpose, CRA maintains a two-byte location counter as a pointer into this area. The initial value of the location counter is zero. As CRA runs and produces output code, it places this code in the output area at the position specified by the location counter, and then advances the location counter past the bytes just inserted. The value of the location counter is also controlled by the CRA directive statements ORG and PAGE (explained later). These statements may be used, for example, to advance the location counter past an area without filling it.

The source programmer may explicitly refer to the current location counter value by use of the character "*" in an expression (see later discussion).

Symbol Table

The most fundamental logical facility provided by an assembler to aid in the output code generating process is its ability to construct and reference a symbol table. A symbol is a sequence of one to six alphanumeric characters beginning with a letter. Each programmer-defined symbol is given an entry in this table where it is assigned a two-byte value which is often an address value equal to the location counter contents at the point where the symbol is "defined" in the program. However, it may also be a user-specified value (for example, a constant) if the symbol is defined in an EQUATE directive (explained later). Thus, while a symbol most often represents an address, it may also represent a specific CPU register, an I/O device number, or any other user-specified value (e.g., a constant, an immediate argument, etc.).

A symbol normally appears at many points in the source program. At one of these points, it is assigned a value in the symbol table; i.e., it is defined. At all other points (whenever the symbol is referenced), this value is used by the assembler to derive or produce code. Thus, by changing the value assigned to a single symbol, the programmer can make substantial changes in his object code file.

A symbol is also called a label, a name, an identifier, or a symbolic address or pointer.

Expression Evaluation

As CRA processes source statements, it produces hexadecimal code values. Much of this code is the direct equivalent of explicit, user-specified information in the source program (constants, register or device numbers, operation code mnemonics). Other code values are derived indirectly using either the current value in the location counter or the value assigned to a specific symbol in the symbol table. The code values thus produced are either assembled into the output stream, as previously described, or assigned to new symbol table entries (when new symbols are defined).

At various places in the source program, CRA will be expecting to encounter an expression. An expression is defined as one of the following forms.

1) expression constant	#2F or 47
2) *	*
3) *±expression constant	*+47 or *-#F1
4) symbol	SAM or J65MP
5) symbol ±expression constant	AREA9-#2F

where an expression constant contains an explicit hexadecimal or decimal value. (The acceptable forms for expression constants are described later.) CRA evaluates the expression by using the explicit constant value (if present), the current location counter value (for the "*", if it is present), the symbol's value in the symbol table (if a symbol is present), and by performing the required arithmetic operation (if necessary). The result is always a two-byte value which may be disposed of in any one of several ways, as discussed further later. Spaces adjacent to the + or - operators are optional.

Following is a more detailed definition of the syntax which CRA is designed to recognize.

COSMAC Level I Assembly Language

Lines and Comments▲

Each line or record in the source file is distinguished by an ending carriage return character. A **line** may consist only of a comment or of one or more **statements** optionally followed by a comment.

A **comment** is any series of characters beginning with two periods. It extends to the end of the line. Thus, the occurrence of two periods at any point in a line causes CRA to ignore the remainder of the line. Statements within a line are normally separated by semicolons (with the last statement optionally terminated by a semicolon). Within each statement, spaces (blanks) may be used freely (except within symbolics and mnemonics) in order to improve readability. CRA will ignore them.

In all the examples which follow, a pair of square brackets will be used to enclose an optional entity—one which may or may not be included. Examples of valid lines are then:

1) ..COMMENT
2) STATEMENT$_1$[;STATEMENT$_2$; ...;
   STATEMENT$_n$][;] [...COMMENT]

Symbol Definitions

(Statement Labels and Equate Statements)

Any statement may optionally begin with a symbol (called a "statement label") immediately followed by a colon. Under these conditions, the symbol is entered into the symbol table and assigned the present location counter value. A statement thus has the form:

[SYMBOL:] STATEMENT BODY

(For example, LOOP: INC R4)

A symbol is also defined when it appears as the left-hand part of an EQUATE statement, which as the form:

SYMBOL=EXPRESSION
(For example, READER=6)

In this case, the expression is evaluated and the resulting two-byte value is assigned to the symbol in the symbol table. (Acceptable forms for symbols and expressions have already been explained.)

Thus, a symbol definition is indicated to CRA by the occurrence of ":" or "=" immediately after a leading sequence of alphanumeric characters in a statement.

When a symbol is equated to a register number, only a decimal or a hexadecimal number should appear on the right side of the equation. For example:

COUNTR=7 is correct
COUNTR=#07 is correct
COUNTR=R7 is incorrect
DELAY=COUNTR is also correct

Explicit Constants

At numerous points in the source program, the programmer desires to directly specify **explicit constants** to CRA. Most often (but not always) the hexadecimal equivalent of an explicit constant is inserted directly into the output code stream at the point where it appears in the source program. (For example, initial data values and immediate arguments may be explicitly defined this way.) CRA allows the programmer the ability to specify absolute constants in binary, hexadecimal, decimal, and alphanumeric forms. The possible explicit constants are summarized below.

Hexadecimal constants: A hex constant is specified with either of the following forms:

Example:

1) #hh...hh	**#3E0F**
2) X'hh..hh'	**X'3E0F'**

where each h is a hex digit (0 to F). **CRA requires that an even number of hex digits be specified.** There are further restrictions on hex constant lengths under certain conditions.

Decimal constants: A decimal constant is specified with either of the following forms:

Example:

1) dd...dd	**635**
2) D'dd...dd'	**D'635'**

where each d is a decimal digit (0 to 9). Each such constant is converted into hex, producing one or two bytes, depending on the space required to represent it. Decimal values greater than 65535 are converted to hex but then truncated to two bytes (upper bytes removed).

Expression constants: An expression constant may be either form of the hex constant or the first form of the decimal constant. Because an expression translates to two bytes, a hex expression constant should normally be restricted to two or four digits in length.

Binary constants: A binary constant is specified in the form:

B'bb...bb'
(For example, B'01101')

where each b is 0 or 1. Up to eight bits may be specified. Each such constant is converted to one byte, with leading 0's assumed.

▲NOTE: All discussion regarding special CRA punctuation characters (such as semicolon, colon, period, asterisk, parenthesis, equals sign, number sign, apostrophe, etc.) refer to those which do not appear within text constants (defined later). Any character within a text constant has no special punctuation significance to CRA.

Text constants: A text constant is specified using the form:

T'cc...cc'

(For example, T'THIS IS TEXT')

where each c is any printable character, including space. Each character is converted to its ASCII code equivalent (see Appendix F) and is represented in one byte. Characters that have no graphic associated with them (i.e., ETX, DC-3, CR, LF, etc) should not be used within a text constant. Entering an apostrophe within a text constant is treated specially, however. See "Additional Notes" below.

Address Constants

The programmer finds it useful to specify not only explicit or absolute constants, but also derived constants whose values are assigned or "computed" by the assembler. Because the fundamental function of the assembler is to assign address values, such constants are normally called address constants. For CRA, an address constant has one of the following forms:

Examples:

1) A(expression) **A(GEORGE + 2)**
2) A.1 (expression) **A.1(LOOP)**
3) A.0(expression) **A.0(*-X'10')**

where the permissible forms for an expression have already been defined. For all cases, the resulting constant is derived by first evaluating the expression. In the first case, the two-byte result is the constant. In the second case, only the upper byte is used; for the third case, only the lower byte. For all cases, the resulting one- or two-byte value is assembled directly into the code output.

Operation Mnemonics

CRA uses special two-, three-, and four-character mnemonics to represent the various instructions in the COSMAC instruction set. These mnemonics are listed in Appendix E. When CRA determines that an operation is being specified, it looks it up in a table to determine the code equivalent of the mnemonic. (Note that this table is not the symbol table, which contains only programmer-defined symbols.) Thus, use of an operation mnemonic effectively defines an explicit hex code value to be inserted into the object stream.

Instructions and Operands

There are two types of output code-producing statements: **instructions** and **data lists.** An instruction begins with an instruction operation mnemonic. In some cases (such as IDL, RET, LDX, etc.) this mnemonic is all that needs to be specified. In most cases, however, the operation mnemonic must be followed by an operand. The form of the operand (i.e., the additional information which the programmer needs to supply to fully define the instruction) depends on the type of instruction. The four operand forms follow.

Register operands: Many instructions (e.g., INC, LDA, etc.) include a hex digit identifying one of the scratchpad registers. The operand field in such a statement may include either a single hex digit, or a symbol. For the last case, CRA uses the least significant hex digit of the symbol's value in the symbol table as the register identifying field.

Examples:

```
DEC 9
LDA RF
PLO SAVE
```

("SAVE=0F" could have previously defined SAVE.)

I/O device operands. The instructions OUT and INP require a device-identifying field. The operand in such a statement may be a single digit in the range 1 to 7, or a symbol. Again, for the latter case, a symbol table lookup occurs, using the least significant hex digit of the symbol's value (checking also that it is within the appropriate range).

Examples:

```
OUT 4
INP READER
```

Branch addresses: Every branch instruction requires an operand specifying the branch address. If the mnemonic is a short branch, a one-byte operand is generated. A two-byte operand is generated if the mnemonic is a long branch. Whenever CRA sees a branch operation mnemonic, it expects to next find an operand in the form of an expression. The acceptable forms for expressions have already been defined. In case of a short branch, CRA evaluates the expression by getting a two-byte address, checks that this address is within the current 256-byte page by examining the upper byte, and uses the lower byte as the second byte in the instruction. For a long branch, the upper byte represents the page number, and the lower byte is the address within that page.

Examples:

If **A(LABEL)** is **#6789**
BZ LABEL
generates **#3289**
and **LBZ LABEL**
generates **#C26789**

Immediate operands: Several instructions include a second byte as an immediate argument. The operand field in such a statement may be any one-byte constant (i.e., an absolute or explicit constant or an address constant) or a symbol. For the latter case, CRA uses the least significant byte of the symbol's assigned value.

Examples:

```
XRI X'FF'
ADI INCREM
LDI A.0(*)
```

Note: When an immediate argument is specified, it is the programmer's responsibility to make sure that it is

a **one-byte** constant. If it is longer, CRA will not generate an error message, but will merely insert the entire constant into the output stream, possibly causing an error during program execution.

Data Lists

The typical program normally includes memory areas which contain data values. Statements which define initial data values are also code-producing statements (although the code generated is normally not "executable"). The data list is a special statement provided for these purposes. It begins with either a comma or the special mnemonic "DC" (which stands for "Define Constant") and is followed by a sequence of one or more constants separated by commas. Each constant may be an absolute, explicit constant (hexadecimal, binary, decimal, or text) or an address constant or a symbol. For the last case, to be consistent with the treatment of symbols as immediate data, CRA substitutes the lower byte of the symbol's assigned value. Thus, a constant in a data list is similar to an immediate operand, but now a length greater than one byte is entirely justifiable.

Examples:

```
DC X'ABCD',355
,#ABCDEF,T'TEXT',B'011'
```

Note: Any statement may be directly followed by a data list without the intervening semicolon. For example LDA 9,#3001.)

CRA Directives

The EQUATE DIRECTIVE (of the form SYMBOL=EXPRESSION) has already been discussed. Three other directive statements are also recognized by CRA:

ORG Statement: This statement is written "ORG" followed by an expression. CRA executes this directive by setting the location counter equal to the value of the expression.

Example: ORG*+20 ..Reserve 20_{10}bytes of space

PAGE statement: The PAGE directive, simply written "PAGE", increases the value of the location counter to that of the beginning of the next 256-byte page; i.e., the upper byte of the location counter is incremented and the lower byte is set equal to zero.

END statement: The END directive, written "END", informs CRA to terminate the assembly. It should appear only once, as the last statement in the source program. The END directive is normally followed by a DC3 character. The DC3 is produced by the EDITOR to signify an end of file.

Thus, in addition to recognizing all the instruction operation mnemonics listed in Appendix E, CRA also recognizes the special mnemonics "DC", "ORG", "PAGE", and "END".

Additional Notes

1) As noted earlier, a space is not permitted within a syntactic entity (symbol, mnemonic, constant, etc.). A space is not permitted between a symbol being defined and the following colon or equals sign. Note, however, that a space within a text constant is permitted. It is translated into its ASCII equivalent code. There is a case where a space is required as a punctuation character. In order to distinguish an operation mnemonic (including ORG) from its following operand (if present), CRA expects to find at least one space.

2) An apostrophe may be included within a text constant by preceding it with a "dummy apostrophe". Thus, the string IT's is written as a text constant as

```
,T'IT''S'
```

3) Special control characters (non-printing characters, such as carriage return, line feed, etc.) should not be placed within the quotes of a text constant. Rather, they should be defined by splitting the text constant into two successive text constants, with the intervening control character represented with a hex constant (using its ASCII code). For example:

```
,T'LINE1',#0D0A,T'LINE2'
```

4) Several COSMAC instructions execute by automatically advancing the pointer to an operand byte after processing it. If the pointer to the operand byte is the same as the current program counter (for example, if X=P or if N=P), then the operand byte may be considered an immediate operand (provided an auto-increment occurs). A statement for such an instruction (under the conditions specified) is most conveniently followed by a comma followed by the one-byte immediate constant. This sequence is permissible because any statement may be immediately followed by a data list - omitting the intervening semicolon.

For example, assuming P=0, the sequence SEX 0; OUT 5,X'52' outputs the immediate hex constant, #52, to output port 5 and continues.

5) In general, any symbol may be referenced before it is defined in a program (termed a "forward reference"). Only one restriction exists: A symbol on the right-hand side of an EQUATE statement (i.e., in the expression) must have been previously defined.

6) CRA uses the location counter value before a statement is processed as the value for any "*" occurring within the statement. Thus, for example, for BR *+3, the value used for the * is the location where the branch byte (hex 30) will be placed, not one byte past that. Thus, BNI * will cause a program loop until flag 1 goes true.

Code Examples and Review

Fig. 9 is a portion of a program illustrating examples of various acceptable CRA statements. It contains a

```
LOCATION  OUTPUT          LINE
COUNTER   CODE            NUMBER      SOURCE LINES                          COMMENTS
0000      ;               0044
0000      7100;           0045        DIS; ,0                               .. DISABLE THE INTERRUPTS
0002      C083F3;         0046        LBR INIT                              .. INITIALIZE THE REGISTERS
0005      ;               0047
0005      D483F0;         0048        SEP CALL; ,A(OSTRNG)
0008      0D0A4241554420; 0049        ,CR,LF,T'BAUD RATE (300 OR 1200)? ',0
000F      52415445202833  ;0049
0016      3030204F522031; 0049
001D      323030293F2000; 0049
0024      F800ADBD;       0050        LDI 0; PLO ASL; PHI ASL               .. CLEAR THE REGISTER
0028      ;               0051
0028      D4813B;         0052  INP1: SEP CALL; ,A(READAH)                  .. GET THE RATE
002B      3328;           0053        BDF INP1
002D      ;               0054
002D      9DFB12;         0055        GHI ASL; XRI #12                      .. IS IT 1200?
0030      3235;           0056        BZ B1200
0032      ;               0057
0032      F814C8;         0058        LDI #14; LSKP                         .. FOR 300 BAUD
0035      ;               0059
0035      F804B0;         0060  B1200: LDI #04; PHI FLAG                    .. FOR 1200 BAUD
0038      ;               0061
0038      D48774;         0062        SEP CALL; ,A(FNDTAP)                  .. GET THE TAPE #
003B      ;               0063
003B      D48502;         0064  MORLST: SEP CALL; ,A(READT)                 .. READ A BYTE FROM TAPE
003E      3BAE;           0065        BNF PROMPT                            .. PARITY ERROR ABORTS
0040      3FAE;           0066        BN4 PROMPT                            .. BREAK KEY ABORTS
0042      ;               0067
0042      9FFB13;         0068  CHKOUT: GHI CHAR; XRI EOF                   .. EOF MEANS WE ARE DONE
0045      32AE;           0069        BZ PROMPT
0047      9FFB0D;         0070        GHI CHAR; XRI CR                      .. FOR CARRIAGE RETURN WE
004A      3251;           0071        BZ RETURN                             .. DO SOMETHING SPECIAL
004C      D40067;         0072        SEP CALL; ,A(PRNTER)                  .. OTHERWISE SEND IT OUT
004F      303B;           0073        BR MORLST                             .. AND GO BACK FOR MORE
0051      ;               0074
```

Fig. 9—CRA Listing of Sample Program

Hex Value	Symbol
0D0A	CRLF
0009	TALLY
0003	LOOP
0016	SAM 1
0007	READER
0005	NUMBER
0002	F
0105	LABEL
0105	RET
0000	

Fig. 10—Symbol table for sample progam of Fig. 9.

listing of the program and the corresponding output code generated. Fig. 10 contains the symbol table for a typical program.

In Fig. 9, the left-hand column gives the location counter value before the line was processed. The next column gives the output hex code generated at that location by the line. (Terminating semicolons in this column should be ignored. They are present to format the output file properly for subsequent loading of the object program. See later operating instructions.) The next column gives a source program line number for reference purposes, and finally the source code is reproduced. The running comments in the source program refer to the statement examples where they appear.

By reading the source program in detail (paying special attention to the running comments), one can quickly review much of what has been said concerning COSMAC Level I Assembly Language. Output code values may be verified by referring to Appendices E and F.

Error Messages

Whenever CRA detects a violation of its **syntax** rules, it generates an error message. There are, however, some possible program errors which will not be detected by CRA because they do not result in syntax rule violations. For example, R3=8; INC R3. R3 is now a symbol, the value of which is 8, so register 8 gets incremented.

When there is a syntactical error, CRA indicates it first, by printing the line in violation using its standard listing format (location counter, output code, line sequence number, source line); second, by inserting a "?" at the detection point in the source line; and third, by printing an error code on the next line. If the error is detected at the end of the line, the "?" may be omitted. In most cases, by looking up the error code meaning in the listing which follows and by noting the position of the inserted "?", the user can easily determine the nature of the error.

It should be emphasized, however, that it is possible that an error at one point in a source line may be interpreted by CRA as an error at a different point. For example, in TTEXT...COMMENT, a single quote is missing after TEXT. It will not be detected until the end of the line. (In fact, if the comment happens to end in a single quote, the error will go undetected.) Further, and more important, it is possible for the error code to indicate one type of error when another actually occurred. For example, the statement SAM INC 3 is missing a colon after the label SAM. The primary meaning of the error code which will return in this case is: "unrecognized mnemonic". This response is understandable because if CRA does not detect a colon or an equals sign, it assumes that the statement does not begin with a symbol. It therefore expects a mnemonic or a comma and does not find either.

An error on one line may cause several lines to be flagged. This response typically occurs when a line containing a label is flagged because of a missing colon and all subsequent references to that label are also noted as "undefined."

Whenever an error exists, the output code is questionable. However, as best it can,CRA increments its location counter past this code and continues to process the source program, possibly detecting further errors which it flags similarly. Detection of an error does not stop assembly of a program. CRA continues in its attempt to find all syntax errors.

The possible error codes and their meanings are given in Table VII, and a summary of error messages is given in Table VIII. If in the process of generating a listing CRA creates a line that exceeds the standard length (typically 78 characters), the line is broken by a (CR) (LF) sequence. The rest of the line is continued on the next line but is preceded by a continuation mark—a period.

TABLE VII—CRA Error Codes and Their Meanings

Error Code	Meaning
01	**UNRECOGNIZED MNEMONIC OR MISSING COMMA** The body of a statement (other than EQUATE) must begin with either a valid operation mnemonic or DC, ORG, PAGE, END, or a comma.
02	**PREVIOUSLY DEFINED SYMBOL** An attempt has been made to define a symbol which already has an entry (and a value) into the symbol table.

TABLE VII—CRA Error Codes and Their Meanings (Con't)

Error Code	Meaning
04	**INVALID CHARACTER WITHIN BINARY CONSTANT** CRA is in the process of evaluating a binary constant and has found a character other than 0/1 or the trailing single quote (which may be missing).
05	**BINARY CONSTANT TOO LONG** The limit is eight bits.
06	**EXPECTED HEX OR DECIMAL CONSTANT HERE** CRA is in the process of evaluating an expression or a constant and expects to see a hex or decimal constant at this point and does not find one. (Note: Under certain conditions, this diagnostic may occur as the result of an undefined symbol.)
07	**UNDEFINED SYMBOL** CRA encounters a symbol reference and wants to use its value, but does not find it listed in the symbol table.
08	**EXPECTED EXPRESSION HERE** CRA determined that an expression was to follow next and did not find leading characters which were proper.
09	**INVALID CHARACTER WITHIN HEX CONSTANT** CRA is in the process of evaluating a hex constant and has found an invalid character. (This error code may be caused by an uneven number of hex digits.)
10	**MISSING TRAILING QUOTE IN TEXT CONSTANT** Note that the error marker "?" will not appear because this error is always detected at the end of a line.
11	**PERIOD ERROR** Either illegal use of a single period or a missing period beginning a comment.
12	**LEADING CHARACTER ERROR** At the beginning of a statement, a leading alphabetic or comma was not found.
14	**BRANCH OUT OF PAGE** A branch address was evaluated and the upper byte did not agree with that of the location counter (see Note 1).
15	**INVALID REGISTER NUMBER** The LDN R0 is an illegal operation. It would otherwise assemble into a hex 00 operation code, the code for an idle instruction.
16	**DEVICE NUMBER OUT OF RANGE** In an OUT or INP instruction, the explicit or symbolic device number had a value greater than 7.
OVFL	**SYMBOL TABLE SPACE EXHAUSTED** The amount of RAM allocated to the symbol table has been used up. The user must remove labels by, for example, using *'s more often or branching relative to existing labels. The program can also be broken up into parts, or more RAM can be added to the CDS. (Note: This error condition halts CRA. A restart of the assembler is required, i.e., $U).

Notes

(1) Although CRA makes every effort to increment the location counter properly when it processes statements that contain errors, it is, of course, possible for the location counter to have a value at any given point different from that which it would have for an error-free program. As a result, it is conceivable that the BRANCH OUT OF PAGE error diagnostic either will occur erroneously or will not be generated when it should be.
(2) The semicolon separating multiple statements in a line is used mainly as a checking device. When CRA has processed an error-free statement and is reinitialized to look at the next one, the intervening semicolon is merely ignored. Thus, it is not really mandatory that the semicolon be used between two statements on a line, if the first is "known" not to contain errors. Clearly, however, it is a necessary practice not only for readability, but also for verification purposes.

TABLE VIII—Summary of CRA Error Messages

Error Code	Meaning
01	UNRECOGNIZED MNEMONIC OR MISSING COMMA
02	PREVIOUSLY DEFINED SYMBOL
04	INVALID CHARACTER WITHIN BINARY CONSTANT
05	BINARY CONSTANT TOO LONG
06	EXPECTED HEX OR DECIMAL CONSTANT HERE
07	UNDEFINED SYMBOL
08	EXPECTED EXPRESSION HERE
09	INVALID CHARACTER WITHIN HEX CONSTANT
10	MISSING TRAILING QUOTE IN TEXT CONSTANT
11	PERIOD ERROR
12	LEADING CHARACTER ERROR
14	BRANCH OUT OF PAGE
15	INVALID REGISTER NUMBER
16	DEVICE NUMBER OUT OF RANGE
OVFL	SYMBOL TABLE SPACE EXHAUSTED

6. Hardware Structure and System Signals

In this chapter, the hardware components of the MCDS are described and the names and functions of the system signals are discussed.

Module Descriptions

The modules included in the MCDS System are described below. The block diagram in Fig. 11 shows the major functions of the CDP18S601 and the CDP18S652 and how they are interconnected.

The **CDP18S601 Microboard Computer** contains a CDP1802 CPU, a crystal-controller clock, read-write memory, parallel I/O ports, a serial communications interface, power on reset, and an expansion interface. Four on-board sockets are provided for read-only memory, enabling the user to select 4 to 8 kilobytes of mask-programmable ROM or PROM, depending on the applications.

The **central processor** for the CDP18S601 is the 8-bit silicon-gate CMOS RCA COSMAC Microprocessor CDP1802. The CDP1802 has 16 general-purpose registers each 16 bits wide. Any one of these registers may be dynamically designated as the program counter, thereby giving the system multiple program states. Each register may also be used for data storage or as memory pointers for subroutines, I/O stacks, and the like. One register each is designated for DMA and Interrupt pointers. The CDP1802 provides a serial data out connection, Q, and four external flag input pins, EF1 through EF4, which may be used as test and branch conditions independently.

Eight MWS5114 RAM's supply the Development System with 4 kilobytes of CMOS **read-write memory.** Four sockets are provided for four or eight kilobytes of nonvolatile **read-only memory.** RCA CDP1834 mask-programmed CMOS ROM's or 2708, 2758, or 2716

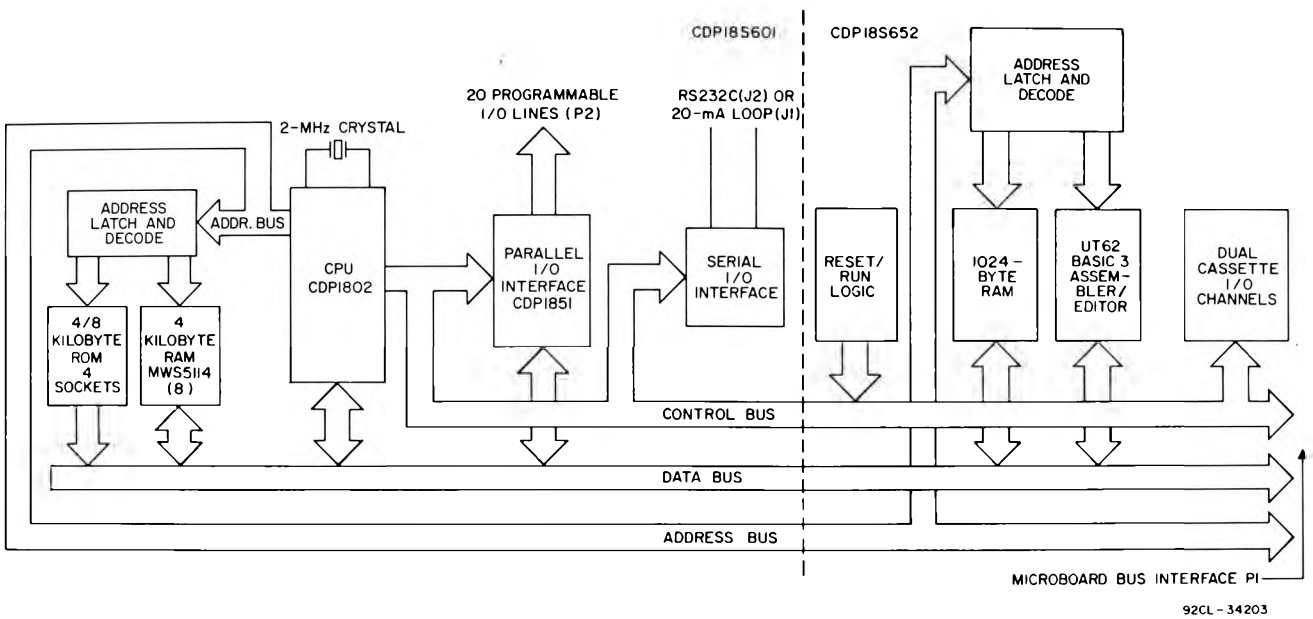


Fig. 11—Block diagram of RCA COSMAC Microboard Computer Development System CDP18S694.

EPROM's may be used in these sockets. Each of these memory types may be placed independently in any of the 16 four-kilobyte blocks comprising the 64-kilobyte system memory space.

With the CMOS programmable I/O Interface CDP1851, the CDP18S601 provides the Prototyping System with 20 programmable I/O lines. The software customizes each of these lines as input, output, bidirectional, or bit-programmable with or without unique handshaking signals for each application. A serial communications interface, with both 20-milliampere loop and EIA RS232C capability, is driven by the Q and EF4 serial I/O lines of the CPU. The baud rate and the data format are determined by software. Edge connectors support the parallel I/O lines and the Microboard Universal Backplane. Right-angle header connections are used with the serial communications interfaces.

The **CDP18S652 Combination Memory and Tape I/O Control Module** contains sockets for 24 kilobytes of ROM addressed in the upper half of memory. Also on board is a 1 kilobyte block of RAM addressed in the memory space 8C00H through 8FFFH. Two tape I/O channels interface to audio cassette tape drives and have software-controlled START/STOP through the "REMOTE" jack. LED's indicate I/O data flow for each channel. A RESET/RUN switch connects to on-board CPU control logic to enable system startup at 8000H (UT62) or 0000H (user memory).

As shipped with the CDP18S693 System, the CDP18S652 Module is augmented to contain the UT62 monitor in ROM and BASIC3 in ROM. With the CDP18S694 System, the Module also contains a text editor and Level I assembler in ROM. Two additional locations are available for two 2-kilobyte or one 4-kilobyte EPROM or ROM.

For more detailed information about the CDP-18S601 or CDP18S652 Microboard Modules, refer to the MB-601 and Appendix G.

The ROM-based Utility Program UT62 operates through any standard data terminal (EIA RS232C or 20-milliampere loop) to allow the user to monitor and alter the contents of memory and to start execution at any address. Operating details for the UT62 are given in Chapter 2.

The **chassis** provided with the CDP18S693 development System can accommodate six Microboard modules in addition to the CDP18S601 and CDP18S652 supplied with the System. Because the chassis utilizes the RCA COSMAC Microboard Universal Backplane (See Table IX for the Backplane Connector Pin List), the Development System can be readily expanded with any of the Microboard

Table IX—Pin Terminals and Signals for the RCA COSMAC Universal Backplane Connector

Wire Side				Component Side			
Pin	Mnemonic	Signal Flow	Description	Pin	Mnemonic	Signal Flow	Description
A	TPA-P	Out	System Timing Pulse 1	1	DMAI-N	In	DMA Input Request
B	TBP-P	Out	System Timing Pulse 2	2	DMAO-N	In	DMA Output
C	DB0-P	In/Out	Data Bus	3	RNU-P	—	Run Utility Request
D	DB1-P	In/Out	Data Bus	4	INT-N	In	Interrupt Request
E	DB2-P	In/Out	Data Bus	5	MRD-N	Out	Memory Read
F	DB3-P	In/Out	Data Bus	6	Q-P	Out	Programmed Output Latch
H	DB4-P	In/Out	Data Bus	7	SC0-P	Out	State Code
J	DB5-P	In/Out	Data Bus	8	SC1-P	Out	State Code
K	DB6-P	In/Out	Data Bus	9	CLEAR-N	In	Clear-Mode Request
L	DB7-P	In/Out	Data Bus	10	WAIT-N	In	Wait-Mode Request
M	A0-P	Out	Multiplexed Address Bus	11	-5V/-15V	—	Auxiliary Power
N	A1-P	Out	Multiplexed Address Bus	12	SPARE	—	Not Assigned
P	A2-P	Out	Multiplexed Address Bus	13	CLOCK OUT	Out	Clock from CPU Osc.
R	A3-P	Out	Multiplexed Address Bus	14	N0-P	Out	I/O Primary Address
S	A4-P	Out	Multiplexed Address Bus	15	N1-P	Out	I/O Primary Address
T	A5-P	Out	Multiplexed Address Bus	16	N2-P	Out	I/O Primary Address
U	A6-P	Out	Multiplexed Address Bus	17	EF1-N	In	External Flag
V	A7-P	Out	Multiplexed Address Bus	18	EF2-N	In	External Flag
W	MWR-N	Out	Memory Write Pulse	19	EF3-N	In	External Flag
X	EF4-N	In	External Flag	20	+12V/+15V	—	Auxiliary Power
Y	+5V	In	+5 volts dc	21	+5V	In	+5 volts dc
Z	GND	In	Digital Ground	22	GND	In	Digital Ground

Milliwatt Computer System Modules or with a user-designed module, as needed. Access holes for the cable connections are provided in the chassis base. In addition, rubber feet are provided, as well as holes for vertical mounting.

The **CDP18S023 Power Converter** is a convenient, compact power supply that plugs into any standard 110-volt 60-Hz wall outlet. It has a regulated output of +5 volts dc ±5% at 600 milliamperes

The CDP18S023 can operate the 20 milliampere loop interface and 5-volt ROM's such as the CDP1834, the 2758, or the 2716. Provision is made for the addition of two auxiliary voltages. Backplane pins 11 and 20 are for a negative and a positive voltage, respectively. For example, +12 volts and -5 volts on these terminals allow the use of 2708 EPROM's as well as the RS232C interface. The provision of +15 volts and -15 volts would allow the use of analog circuits as well as the RS232C interface.

System Signals

The CDP18S693 and CDP18S694 make use of the following conventions for designating the system signals. The designation for each signal may be followed by a hyphen and either the letter N or the letter P. The suffix N means that the signal is asserted (true) when the wire is at ground. The suffix P means that the signal is asserted (true) when that wire is at the highest logic level (+5 V). Thus, the signal name gives the meaning assigned to a specific conductor, and the suffix defines the electrical value of the asserted (true) state. An alternative convention uses a barred signal (e.g. $\overline{EF1}$) to indicate that the signal is asserted (true) when the wire is at ground.

The signals on the universal backplane and their pin assignments are listed in Table IX. The majority of these signals is derived from the CDP1802 Microprocessor; the signals are described below. For additional information, refer to the **User Manual for the CDP1802 Microprocessor,** MPM-201 and to the data sheet for the CDP1802, File No. 1023.

TPA, TPB—Timing pulses generated by the CPU which occur once in each machine cycle. TPA trailing edge is used to latch the high-order memory address. TPB trailing edge is used to latch output data from the data bus.

DB7 through DB0—Eight bidirectional data bus lines. Taken directly from the CPU bus pins, these lines transfer data among the memory, CPU, and I/O devices.

A7 through A0—Eight memory address lines from the CPU. The 16 memory address bits are multiplexed on this address bus. The high-order eight bits are presented early in each machine cycle and must be latched at the TPA trailing edge. The CDP18S601 buffers, latches, and decodes these bits for the onboard memories. Any external memory must provide its own latches. During the latter part of the cycle, the low-order eight bits are presented on this address bus and need not be latched.

$\overline{MWR}$—A WRITE command from the CPU to the memories. Address lines are stable at this time. Actual writing or latching occurs at the trailing edge.

$\overline{MRD}$—A READ command from the CPU to the memories and a direction indicator for I/O data transfers. In the I/O instructions it corresponds to N3 (N register, internal to the CPU) which distinguishes I/O inputs from outputs. MRD must be used to condition output drivers in all memory components, or their output buffers, to avoid contention on the data bus. The absence of MWR must not be interpreted as a READ. Early in a write cycle, data are being driven onto the data bus by the CPU or an input device. If a memory allows its outputs to be enabled while MRD is false before MWR appears, bus contention will occur resulting in unnecessary power dissipation and perhaps circuit failures. Operation using the Micromonitor CDP18S030 is impossible unless MRD is properly used to condition data output.

Q—A single-bit output from the CPU. This bit is set or reset by SEQ (7B) or REQ (7A) instructions. The CDP18S601 may use Q as a serial data output to the RS232C and 20-mA data terminal drivers. It is also available for other uses through the Microboard Bus (P1) and Parallel I/O (P2) connectors. Q may also be tested with a branch instruction and thereby operates as a program switch.

N0, N1, N2—taken directly from the CPU pins, these lines indicate an I/O instruction is being executed. They are derived from the low-order three bits of the N register during an I/O instruction execution only. They are low (false) at all other times. These bits form the primary address identifying the I/O device. Direction of transfer, derived from N3 internal to the CPU, is presented on the $\overline{MRD}$ line. When high, $\overline{MRD}$ indicates data transfer from I/O to memory; when low, from memory, to I/O.

$\overline{EF1}$, $\overline{EF2}$, $\overline{EF3}$, $\overline{EF4}$—Taken directly to the CPU pins, these inputs can be tested by conditional branch instructions. The CDP18S601 uses EF1 and EF2, conditioned by the secondary I/O address, to test the READY state of I/O ports A and B. The CDP18S652 uses conditioned EF1 to read serial data from the cassette tape interface. The serial data interface input is presented directly on EF4 or EF3 chosen by link LK36. I/O devices using the INT line may make use of the EF lines to identify the device. They may also be used to indicate priority or status.

$\overline{INT}$—Taken directly to the CPU pin, the interrupt line causes a transfer of control from the current program counter to register 1. Interrupts may be inhibited by software. If Interrupt Enable (IE) is set, recognition of INT results in completion of execution of the current instruction, followed by an S3 machine state during which designators X and P are stored in T. Then, X is set to 2, P is set to 1, and IE is reset to 0. The S3 state lasts one machine cycle (eight clocks), after which processing resumes with R1 as program counter.

$\overline{DMAI}$, $\overline{DMAO}$—Taken directly to the CPU pins and not utilized by the CDP18S601, these lines allow offboard I/O controllers rapid direct memory access.

The CPU monitors these data transfers, going into an S2 machine state for each byte transfer. R0 is used as the memory pointer and is automatically incremented each time. Thus, DMA transfers are interleaved with normal processing and no software action is required except to initialize R0 before transfer starts. INT and/or an EF may be used to notify the program that a block DMA transfer is completed so that initialization and processing of the data block may be performed. The DMA inputs may be maintained in the true state for contiguous S2 states for the most rapid transfer. In the usual case, however, the DMA request is removed at the TPA of the S2 cycle to obtain a single byte transfer, allowing time for normal processing and for setting up the next byte in the requesting controller. Each S2 state is eight clock cycles in duration.

SC1, SC0—State code outputs from the CPU which identify the type of machine cycle in progress.

	State Code Lines	
State Type	SC1	SC0
S0 (Fetch)	L	L
S1 (Execute)	L	H
S2 (DMA)	H	L
S3 (Interrupt)	H	H

CLOCK OUT—A 2-MHz square-wave clock provided for general use. It is derived from the crystal-controlled oscillator in the CPU.

WAIT, CLEAR—Two control inputs to the CPU which determine the mode of operation.

CLEAR	**WAIT**	**Mode**
L	L	Load
L	H	Reset
H	L	Pause
H	H	Run

The functions of the modes are defined as follows:

Load Mode. Holds the CPU in the IDLE state and allows an I/O device to load the memory without the need for a "bootstrap" loader. It modifies the IDLE condition so that termination of the DMA-IN operation does not force execution of the next instruction. DMA in requests then load memory starting from location zero for as many bytes as there are DMA-IN requests.

Reset Mode. Registers I, N, and Q are reset, IE is set, and 0's (V_{SS}) are placed on the data bus. TPA and TPB are suppressed while reset is held and the CPU is placed in S1. The first machine cycle after termination of reset is an initialization cycle which requires 9 clock pulses. During this cycle the CPU remains in S1, and registers X, P, and R0 are reset. Interrupt and DMA servicing are suppressed during the initialization cycle. The next cycle is an S0 or an S2, but never an S3. Power-up reset is obtained by a Schmitt-trigger buffered RC network connected to CLEAR.

Pause Mode. Stops the internal CPU timing generator on the first high-to-low transition of the input clock. The oscillator continues to operate, but subsequent clock transitions are ignored.

Run Mode. May be initiated from the Pause or Reset Mode functions. If initiated from Pause, the CPU resumes operation on the first high-to-low transition of the input clock. If initiated from Reset, the first machine cycle following Reset is always the initialization cycle. The initialization cycle is then followed by a DMA (S2) cycle or fetch (S0) from location 0000 in memory.

RNU—Run Utility Software. A signal supplied to the system to force the most significant address line true. As a result, the program start is at memory location 8000 instead of 0000. The CDP18S652 Microboard provides the Reset and Run Modes for the System.

7. Memory Addressing and Expansion

The user memory provided with the MCDS CDP18S693 or CDP18S694 is located on the CDP18S601 Microboard Computer. The information in this Chapter on memory addressing and expansion refers to that module.

On-Board Memory Addressing

The high-order eight memory address bits are latched, decoded, and used for generating chip selects for on-board memories. A system of links is provided for placing RAM or ROM in the desired area of the 64-kilobyte address space. Links (wire jumpers) are to be installed as described below. As an alternative, DIP switches may be readily installed in place of the links because the links are arranged in standard 16-pin DIP dimensions.

RAM Address

The RAM on the CDP18S601 is 4 kilobytes of contiguous memory. The high-order four bits of memory address are latched and decoded, and a set of eight links is provided so that any value of the four high-order bits may be selected as the address of this RAM. Thus, the RAM may occupy any even 4-kilobyte block in the memory space.

To set up the RAM address, install two jumpers in link LK11, according to Table X. Alternatively, a DIP switch may be installed if frequent changes are anticipated.

ROM Address

Four 24-pin sockets are provided for user-programmed ROM's. Four ROM types are suitable: CDP1834 (1 kilobyte), 2708 (1 kilobyte), 2758 (1 kilobyte), and 2716 (2 kilobytes, Intel pin-out). The CDP1834 mask-programmable ROM can be used in combination with any of the other three types. No other combination may be used. One to four ROM's may be used.

Two types of links are provided and must be made up by the user to suit the particular ROM configuration selected. The first link type is for accommodating the type of ROM selected (CDP1834, 2708, 2758, or 2716. The second link type is for selecting the memory address space to be occupied by the ROM.

Table X—4-Kilobyte Link Connections

4-Kilobyte Address Space	LINK LK10, LK11, or LK22 Pin Connections
0000-0FFF	1:16, 5:12°
1000-1FFF	1:16, 6:11†
2000-2FFF	1:16, 7:10
3000-3FFF	1:16, 8:9
4000-4FFF	2:15, 5:12
5000-5FFF	2:15, 6:11
6000-6FFF	2:15, 7:10
7000-7FFF	2:15, 8:9
8000-8FFF	3:14, 5:12
9000-9FFF	3:14, 6:11
A000-AFFF	3:14, 7:10
B000-BFFF	3:14, 8:9
C000-CFFF	4:13, 5:12
D000-DFFF	4:13, 6:11
E000-EFFF	4:13, 7:10
F000-FFFF	4:13, 8:9

LINK 11 is associated with the 4-kilobyte RAM.
LINK 10 is associated with the ROM sockets 25 and 24.
LINK 22 is associated with the ROM sockets 27 and 26.
°Preprinted on LK10 (ROM) and on LK11 (RAM)
†Preprinted on LK22 (ROM)

Line LK4 is an 18-pin dual-in-line arrangement with preprinted links to accommodate the 2716 ROM. Table XI gives the connections required for each ROM type.

Links LK10 and LK22 are 16-pin dual-in-line arrangements prelinked to address 0000. A DIP switch may be installed if frequent address changes are expected. Link LK10 provides the high-order four address bits decoded so that two links or jumpers place sockets XU24 and XU25 in any 4-kilobyte block within the 64-kilobyte memory address space. Link LK22 does the same for sockets XU26 and XU27. See Table X for address map and link connections.

Table XI—ROM Type Selection Links

Link LK4 Pins	ROM Type CDP1834	2708†	2758	2716*
1:18	X	OPEN	SHORTED	SHORTED
2:17	X	SHORTED	OPEN	OPEN
3:16	SHORTED	SHORTED	SHORTED	OPEN
4:15	OPEN	OPEN	OPEN	SHORTED
5:14	OPEN	OPEN	OPEN	SHORTED
6:13	SHORTED	SHORTED	SHORTED	OPEN
7:12	X	SHORTED	OPEN	OPEN
8:11	X	OPEN	OPEN	SHORTED
9:10	X	OPEN	SHORTED	OPEN

X=don't care;
*Link LK4 is prewired to accept 2716.
†Use of 2708's requires extra voltages of 12 volts and 5 volts on pins 20 and 11, respectively, of the Universal Backplane Connector P1

To avoid having floating inputs to the gates, both links LK10 and LK22 should always have two jumpers. For example, if sockets XU26 and XU27 are unused, LK22 may be jumpered the same as LK10. Otherwise, spurious chip selects may be generated, turning on the three-state data buffers and causing interference with normal processing.

For 1-kilobyte ROM's such as the CDP1834, 2708, or 2758, links LK10 and LK22 should be jumpered identically in accordance with Table XI. Then, the ROM's should be installed in sockets XU25, XU27, XU24, and XU26, in that order, starting with the lowest-address ROM.

For 2-kilobyte ROM's (2716), links LK10 and LK22 should be jumpered independently in accordance with Table X for the required two 4-kilobyte blocks.Then, socket XU25 is the low 2 kilobytes and socket XU24 is the high 2 kilobytes of the 4-kilobyte block as set in LK10. Similarly, socket XU2 is the low 2 kilobytes and socket XU26 is the high 2 kilobytes of the 4-kilobyte block set in LK22.

One-kilobyte ROM type CDP1834 is the only one that may be used in combination with two-kilobyte ROM's type 2716. If all links are set up for the 2-kilobyte ROM's as shown in Table X for LK4, and LK10 and LK22 are set up for different 4-kilobyte blocks, then a 1-kilobyte ROM in socket XU25 will occupy the two lower 1-kilobyte segments of the 4-kilobyte block. In other words, its 1 kilobyte will "wrap" through the lower 2 kilobytes of the 4-kilobyte block. If it is in socket XU24, it will wrap through the upper 2 kilobytes of the 4-kilobyte block. A 2-kilobyte ROM may be placed in either socket XU24 or socket XU25 while the other is occupied by a 1-kilobyte ROM. Socket XU27 (low 2 kilobytes) and socket XU26 (high 2 kilobytes) may be used in the same manner.

Memory Expansion

The addressable memory range of the CDP18S693 and CDP18S694 is 65,536 bytes. Because of the Microboard Universal Bus and Backplane, the memory provided with the System can be readily expanded by the addition of RCA Microboard Memory Modules or by the use of custom designed memories. A system memory map is given in Fig. 12.

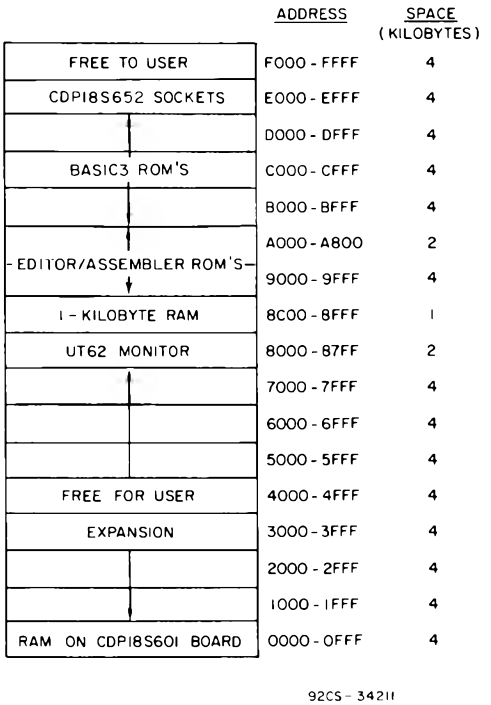


Fig. 12—Memory map for MCDS CDP18S693 and CDP18S694.

RCA Memory Modules

The RCA Microboard Memory Modules available to expand the RAM/ROM capability of the CDP18S693 and CDP18S694 include the following:

4-Kilobyte RAM CDP18S620
4-Kilobyte Battery-Backup RAM CDP18S624
8-Kilobyte RAM CDP18S623
8-Kilobyte Battery-Backup RAM CDP18S622
16-Kilobyte RAM CDP18S621
32-Kilobyte RAM CDP18S629
4-Kilobyte EPROM CDP18S627
8/16/32-Kilobyte ROM/PROM CDP18S625
32/64-Kilobyte EPROM/ROM/RAM CDP18S626

Each of these Microboard modules has decoders with link options that allow the memory to be user-assigned within the 64-kilobyte address range.

Custom Memory Modules

Users who wish to design and construct their own memory systems can do so by carefully observing the physical and electronic constraints of the Microboard

system. The boards use the standard 44-pin, 0.156-inch center edge connector. The connectors in the backplane are on 0.6-inch centers. The boards used for a custom design must be 4.5 inches wide by about 7.5 inches. The Microboard Breadboard CDP18S659 fits into the eight-card chassis and can accommodate up to 40 integrated circuits, thus lending itself readily to custom-designed memory modules.

To optimize system performance, all appropriate CPU signals are wired directly from the CDP1802 to the backplane. When additional modules are used, the capacitive loading on all signals should be minimized by the use of input and output buffers such as types CD4049, CD4050, CDP1856, CDP1857, or the like. In addition, each memory module must latch the high-order address bits with the TPA trailing edge and provide the decoding for memory device selection. The designer should refer to the data booklet for the CDP1802 (File No. 1023) for timing information. The timing information should be carefully adjusted to allow for added loading and circuit delays attributable to the expansion modules.

8. Input/Output Interfacing

One of the fundamental advantages of the COSMAC architecture is the richness of the CPU-I/O interface. A significant number of interfacing resources (many I/O-oriented signals with many different functions) are available for use. These resources include the DMA and INTERRUPT request lines, the four EXTERNAL FLAG input lines, the three N output lines, the software-controlled Q output line, the related control signals SC0, SC1, TPA, and TPB, and, of course, the I/O data bus.

Because the input/output instructions transfer data directly between memory and the I/O device, rather than through an intermediate accumulator register, the need for an additional load or store instruction is eliminated. The data of an I/O read instruction, however, also enters the accumulator register D, at no cost in time, from which it may be tested for a zero value without the need for an intervening ALU operation, or it may be used in any of the many register D operations. A wide variety of interfacing techniques is possible, limited only by the ingenuity and imagination of the designer. A few interfacing techniques are discussed below, each supported by an illustrative example.

Two-Level I/O Interfacing

During an I/O instruction, the CPU presents the low-order three bits of its N register on the N2, N1, and N0 lines. N3 generates the MRD signal to indicate the direction of data flow. Thus, the instructions 61 through 67 and 69 through 6F provide seven output and seven input commands. These instructions may be interpreted by the system as either different commands to the same I/O device or as I/O commands to different devices as addressed by the N lines.

In a larger system more addresses are needed. This requirement can be met by the establishment of a two-level I/O system. One of the output instructions is dedicated to a group-select function. The output byte is latched in the external hardware and may be either linear or binary encoded. The I/O addresses are effectively increased to 96 and 1530, respectively. One of the input instructions is also reserved for reading back to the CPU the latched group number.

In the Microboard system the following conventions are established:

- The **61 output instruction** is used to transmit the group number. The output data byte is latched and decoded by any Microboard in the system having an I/O function. Any I/O function is assigned to a group number and only responds when its group number and its appropriate N register code are transmitted.
- The group number is divided into two parts, the lower four bits being a one-of-four encoding and the high four bits being binary encoded. Thus, the number of addresses provided is 15 binary-encoded plus 4 individual lines, times the 6 commands left after reserving the 61 and 69. The total number of useful I/O addresses is 114.
- The **69 input instruction** is reserved for reading the latched output of the 61 instruction.

The use of the two halves of the group number must be independent and exclusive. That is; the high-order bits must be zero when any low-order bit is used, and the low-order bits must be zero when the high-order bits are used. Once a group number is set up, subsequent 62-through-67 and 6A-through-6F instructions are recognized only by devices assigned to that group number.

The CDP18S601 uses bit three as the group select; that is, the group number $(08)_{16}$ or $(0000\ 1000)_2$ is transmitted by the 61 output instruction to select the programmable I/O port on board.

The CDP18S652 uses group number $(02)_{16}$ or $(0000\ 0010)_2$ to select the cassette I/O channels.

Serial I/O Interfacing

A serial communications interface on the CDP-18S601 provided with both 20-milliampere and EIA RS232C capability is driven by the Q and EF4 serial I/O lines of the CPU. The baud rate and the data format are determined by software (UT62). Electrical interfaces for the 20-milliampere loop and the RS232C data terminals are provided on connectors J1 and J2 respectively. Output drivers are separate, but the input receiver is shared. The only modification

required for the RS232C interface is the installation of a jumper wire in the C5 holes on the CDP18S601 Microboard. The pin connections for connectors J1 and J2 are shown in Table XII.

Serial data input is presented to either EF3 or EF4, selectable by links as shown in Table XIII. UT62 requires the use of EF4. EF3 is made available for user software. The serial interface is always selected and not governed by the two-level I/O conventions.

Table XII—Serial Interface Signals and Pin Connections (J1 and J2)

Microboard Computer 20-mA Serial Interface (J1)

Pin	Signal	Pin	Signal
1	VACANT (KEY)	6	NC
2	NC	7	DATA OUT SOURCE
3	DATA OUT RETURN	8	DATA IN SOURCE
4	DATA IN RETURN	9	NC
5	NC	10	NC

Microboard Computer EIA RS232C Serial Interface (J2)

Pin	Signal	Pin	Signal
1	GND	6	HIGH LEVEL
2	DATA IN	7	HIGH LEVEL
3	DATA OUT	8	HIGH LEVEL
4	NC	9	NC
5	VACANT (KEY)	10	GND

Table XIII—Link LK36 Selection Chart

Link LK36	Function
7:10	Data to EF3
8:9	Data to EF4

Parallel I/O Interfacing

The parallel I/O interface consists of 20 lines provided on connector P2 of the CDP18S601. See Table XIV for the pin connections and signal names. These 20 lines are generated by the CDP1851 Programmable I/O Interface and may be programmed as input, output, or bidirectional individually or as a block. The P2 connector also provides the Q line, EF1 through EF4, CLEAR, three different voltages, and a logic ground.

For more detailed information on the Programmable I/O Interface CDP1851, refer to the data sheet for that device (File No. 1056).

The CDP1851 is assigned to I/O group eight. Therefore, in order to enable access, a 61 output instruction with data = 08 is required before read, write, or control I/O may be performed.

Signal ARDY conditioned by the group select generates EF1; BRDY and group select generates EF1.

Table XIV—Parallel I/O Interface Signals and Pin Connections (P2)

Pin	Signal	Pin	Signal
1	B2-P	2	GND
3	B1-P	4	B3-P
5	B0-P	6	B4-P
7	BSTB-P	8	B5-P
9	BRDY-P	10	B6-P
11	AD7-P	12	B7-P
13	AD6-P	14	GND
15	AD5-P	16	CLEAR-N
17	AD4-P	18	GND
19	AD3-P	20	Q-P
21	AD2-P	22	GND
23	AD1-P	24	EF4-N
25	AD0-P	26	EF3-N
27	ASTB-P	28	GND
29	ARDY-P	30	+5V
31	EF2-N	32	-5V/-15V
33	EF1-N	34	+12V/+15V

BRDY and group select generates EF2. Pins A and B of link LK41 may be jumpered if interrupt-driven software is to be used. Then, INTA or INTB generates INT unconditionally.

Once the group select is accomplished, N1 and N2 are used to address the CDP1851. The following read and write instructions are used to access data, status, and command registers.

62—Write to control register
64—Write to Port A data register (if A is an output)
66—Write to Port B data register (if B is an output)
6A—Read status register
6C—Read Port A data register (if A is an input)
6E—Read Port B data register (if B is an output)

Using the READY Lines for Data Synchronization

The Port A and Port B RDY lines are presented to the CPU EF1 and EF2 lines when the group select is set. Note that there is a logic reversal: when RDY is true, the EF is false. A test for ARDY true might use the B1 instruction (34) which would take the branch if ARDY were false. Even though these RDY lines are primarily intended for "handshaking" with the device on the other end of the cable, they are useful for synchronizing data transfer between the CDP1851 and the CPU.

When a port designated as an **output** port is loaded, RDY goes true. When the receiving device takes the data, it transmits STB which removes RDY. The software can then test RDY until it is false (EF1 or EF2 true), and load the next output byte. When a port is designated as an **input** port, reading the data sets RDY, and the transmitting device resets RDY when it transmits data and STB. Again, the software tests to see if RDY is false and then reads the input byte. **In this case, a dummy read after reset is necessary to raise the first RDY.**

Note that if the remote device is passive, such as a display or a set of points, handshaking is not necessary. The output port may be loaded at any time to change data without acknowledgment from the remote device. Similarly, the input port may be read at any time to store the current state of the input lines.

Using the INTERRUPT Lines for Data Synchronization

If link LK41 A:B is jumpered, INTA and INTB generate INT to the CPU. INT is not conditioned by the group select. INT is set by the remote device sending STB to acknowledge an output port and is reset by loading an output port. Similarly, INT is set by the remote device sending STB to load an input port and is reset by reading the input port. Table XV summarizes the actions of READY and INT for input and output modes.

The software can find the source of the interrupt by setting the group select 08, and then, either testing the RDY lines or reading the status byte. The low-order two bits of the status byte are: bit 0 = INTB; bit 1 = INTA.

Bidirectional Mode

Port A may be programmed to be directional. In this case, Port B must be programmed to be in the bit-programmable mode, to be described later. In the bidirectional mode, ARDY and ASTB become A INPUT RDY and A INPUT STB; BRDY becomes A OUTPUT RDY, and BSTB becomes A OUTPUT STB. Each of the eight lines AD0—AD7 may transmit data in both directions, using the input handshaking lines to synchronize inputs and the output handshaking lines for the output data. Operation is much the same as for independent input and output ports except that output data is gated into AD0—AD7 only when the OUTPUT STB line is raised. In summary, Port A in the bidirectional mode is an output port and an input port sharing the same eight data lines, each having a set of handshaking lines.

Bit-Programmable Mode

Both Port A and Port B are capable of being programmed to be in the bit programmable mode. Port B must be in this mode if Port A is in the bidirectional mode. In the bit-programmable mode, each line in AD0-AD7 and B0-B7 is programmed to be either input or output. In addition, the handshaking lines are programmed to be input or output lines unless Port A is bidirectional, in which case it uses all four handshaking lines. The handshaking lines, when used as data lines, are accessed by a write control for output lines and read status for input lines. The other eight lines in each port are accessed by the usual read and write data instructions.

Interrupts are generated when an input line goes true except that the former handshaking lines cannot generate interrupts. The bits may be individually masked so as not to generate interrupts. The interrupt control word selects one of the two interrupt rules, AND or OR. The AND rule results in an interrupt only when all unmasked lines are true. The OR rule results in an interrupt when any unmasked line is true. The interrupt control word also defines the input lines as logically true when high or logically true when low.

I/O Expansion

RCA I/O Modules

A number of modules are available in the Microboard product line that allow expansion of the I/O capabilities of the MCDS. Included are:

UART Interface CDP18S641
D/A Converter CDP18S642
A/D Converter CDP18S643
A/D and D/A Converter CDP18S644
Parallel I/O Module CDP18S646
D/A Converter CDP18S647
A/D Converter CDP18S648
Floppy Disk Interface Module CDP18S651
A/D and D/A Converter CDP18S654
D/A Converter CDP18S657
A/D Converter CDP18S658
Combination Memory and I/O Module CDP18S660
Video-Audio-Keyboard Interface CDP18S661
PIO Opto 22 Module Interface CDP18S662

Custom I/O Modules

The constraints on custom memory modules are also applicable to custom I/O modules. The physical size must be compatible if the custom board is to plug into the backplane, and the backplane signals should be buffered. Each module must latch the group-select byte and decode the I/O addresses. The user must

Table XV—READY and INTERRUPT Actions for Input and Output Modes.

		Output Port	Input Port
READY	Set by	Loading data	Reading data
	Reset by	STB leading edge	STB leading edge
INTERRUPT	Set by	STB trailing edge	STB trailing edge
	Reset by	Loading data	Reading data

avoid overlapping I/O addresses because bus contention would result. The use of the external flags, DMA, and the interrupts is discussed in the following section. For additional information, refer to the **User Manual for the CDP1802 COSMAC Microprocessor,** MPM-201

System Signals in I/O Interfacing

External Flags EF1 to EF4

The external flags offer a simple yet powerful input interface to the CDP1802 COSMAC Microprocessor in the computer system module of the MCDS. Means by which a program may test an external flag and branch conditionally on its value have been discussed. The use of a flag as a bit-serial data input port was also described. Note that the serial I/O uses EF4 making it unavailable for other devices unless link connection 9:8 on link LK36 on the CDP18S601 Microboard module is opened. Because transmission gate outputs may be connected as "wire ors", several devices may share a specific external flag when necessary. It should also be noted that all four external flag signals are pulled up to V_{DD} through 22-kilohm resistors on the CDP18S601.

In order for an external flag to play a functional role in a COSMAC-based system, it must be tested by the program at the time when action is required. Further, in programs which incorporate a periodic flag test, there must be some means for the interface logic to sense that the flag stimulus has caused a response. In systems where it is necessary to detect failure or error conditions, one or more flags may be used. If immediate action is required, the flags may be used as a means to vector interrupts.

DMA Input

Assuming only one I/O device needs the DMA input port, the arrangement shown in Fig. 13 is possible. Systems having more than one DMA channel (either 2 or more DMA-IN's or a mixed DMA-IN and DMA-OUT) must use the Interrupt facility to establish vectoring. Notice that if interrupts will not be used in the system of Fig. 13, the SC1-P can be used directly for the output port and signal flip-flop: it does not have to be gated with SC0-P. The MRD-N line is also not strictly necessary unless a mixed DMA-IN/DMAOUT system is being constructed.

DMA Output

Assuming only one device is using the DMA-output channel, the circuit of Fig. 14 can be used. Again, note that multi-channel DMA systems must use the Interrupt facility. The discussion concerning SC1-P and MRD-N in the previous section also applies here. The Control Module uses DMA-OUT to terminate IDLE. Isolation of a user-supplied signal by means of a three-state device, diode, or the like is required on this line.

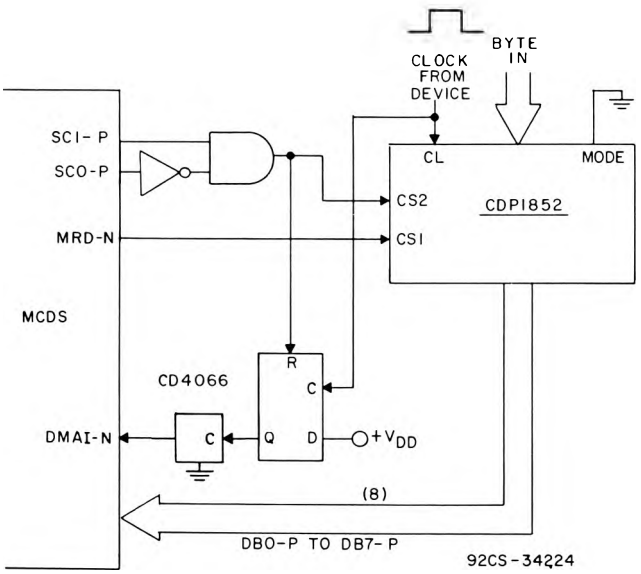


Fig. 13—DMA input example.

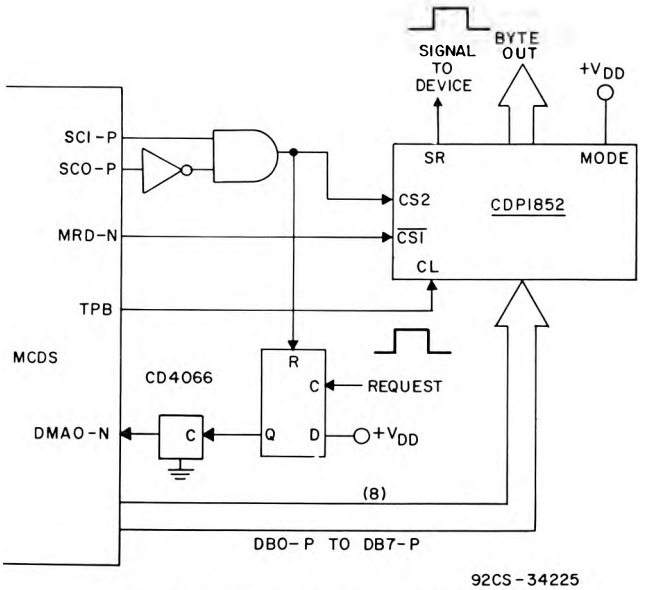


Fig. 14—DMA output example.

Interrupt

A straightforward interrupt implementation is shown in the **User Manual for the CDP1802 COSMAC Microprocessor,** MPM-201.

Systems that require multiple interrupt conditions can be handled in a variety of ways. If the interrupts are synchronous with respect to each other (i.e., there is a prior knowledge that there will be a specific pattern such as ABAB.... or AABCAABCA....), then all the handling can be accomplished with software. The interrupt analysis pointer is merely re-initialized after each service to the address necessary to handle the next service.

The source of an asynchronous interrupt must be deduced from externally generated information, such as the contents of an interrupt status register (interrogated by an I/O READ instruction), any external flag values, or the address of register R0 as dictated by DMA activity. The CPU contains a programmable INTERRUPT ENABLE (IE) bit. Its operation is discussed in detail in the **User Manual for the CDP1802 COSMAC Microprocessor,** MPM-201. Before designing an interrupt-generating I/O circuit for installation in the MCDS, the reader should refer to the material on **"Interrupt and Subroutine Handling"** in the **Instruction Repertoire** Section and the material on **"Interrupt Service"** in the **Programming Techniques** Section of the **User Manual,** MPM-201.

Fig. 15 shows a general purpose interface to the MCDS for multiple interrupts. The interrupt routine would issue a specific input instruction (say '62') and the device requesting the interrupt would put its address on the data bus where it can be examined by the CPU and used to vector to the right routine. Additional hardware would be required if the possibility of simultaneous interrupts exists.

A good programming technique is to put the bytes 71 and 00 a the first two instructions in the program. That will disable interrupts until the programmer is ready for them. At that time, they can be re-enabled by setting X = P, then performing the instruction sequence 70. (X, P) where the byte after 70 contains the initial values for X and P. Refer to the **User Manual** MPM-201 for interrupt servicing program techniques.

Power-On Reset

An RC integrator (R1 and C4 in the control circuit logic diagram) and a Schmitt-trigger circuit (U30) provide a long-time-constant (approximately 150 milliseconds) signal when the +5-volt supply is turned on. This signal appears in the CLEAR-N input to the CPU, the parallel I/O interface, and the I/O group-select latch. After the CLEAR signal, the I/O group select is reset, the parallel I/O interface Ports A and B are set to be input ports, the mask register is reset (monitors all bits), and the status register is reset. The CPU initializes and starts processing at location 0000 (provided the WAIT line is not asserted).

The power-on reset is generated through a transmission gate. External circuits, therefore, may generate CLEAR on P1-9 or P2-16 using transmission gates, three-state, or open-collector devices.

The CDP18S652 provides control of the CLEAR line to the CPU with the RESET/RUN switch.

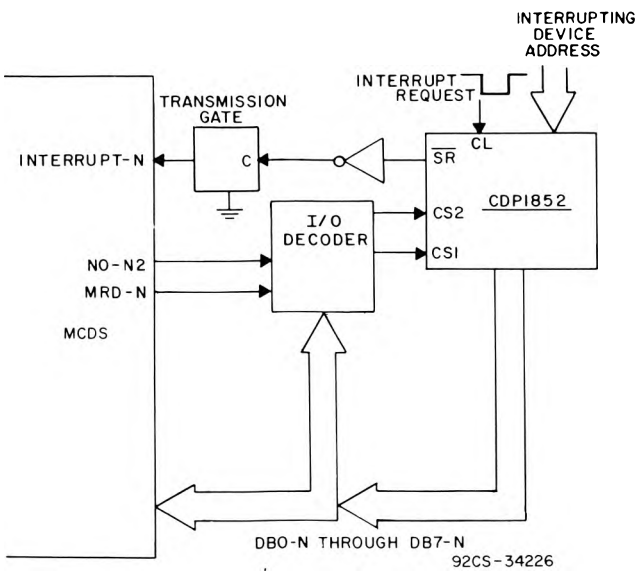


Fig. 15—General-purpose circuit for vectored interrupts.

9. Using the Cassette Tape Drive

The cassette tape interface enables the user to store and retrieve programs or data on low-cost audio cassette tapes. The UT62 monitor program, BASIC3, the Editor/Assembler, and the PROM Programmer all provide for cassette tape data storage. The CDP18S652 Combination Memory and Tape I/O Control Module has two separate cassette tape channels addressed in software as tape 0 and tape 1.

Operation

The UT62 monitor program takes care of tape housekeeping for the user. A tape read routine converts ASCII serial data from tape into 8-bit bytes and stores them in a read block buffer in RAM. Similarly, a write routine fills a write block buffer, then converts the data to serial ASCII, and stores it on tape.

Data on tapes are stored and retrieved in 384-byte (1½-page) blocks. Between blocks, the software has the ability to stop the tape. The Editor/Assembler needs this capability to enable the system to process the data in chunks. The format of data is shown in Fig. 16. A block of data occupies about six seconds of tape.

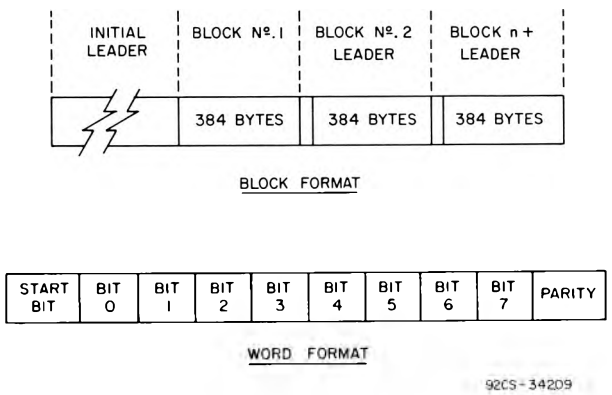


Fig. 16—Data format for cassette tape drive unit.

The first block written stores an extra length of leaders (0's) to provide assurance that the non-oxide portion of the beginning of the tape is skipped. Each byte is serialized into a 10-bit word consisting of a start bit, 8 bits of data, and a parity bit (odd). If a parity error is encountered on tape read during a block, the system issues the message:

PARITY BAD THIS BLOCK

Parity errors can be generated by a dirty or worn tape, dirty tape heads, or improper level settings on the tape drive unit for volume or tone. *The tone control should be set at maximum (10) for best frequency response. The volume control also should be set at 10.*

System Care

At regular intervals the user should *clean the tape heads* with a cotton wetted with alcohol. Also, it is a good precaution to discard worn tapes to avoid damaging the heads. For other tape-handling precautions, see Appendix C.

The recommended cassette tape is Maxell or BASF 60-minute (30 minutes per side) tape or equivalent. Tapes longer than 60 minutes are thin and should be avoided. A 60-minute tape is capable of storing over 115,000 ASCII bytes per side. Tapes of shorter length can be used and, in fact, are desirable for short programs on both sides because they permit fast rewind. Note that one minute of tape (after the extended leader for block 1) can store about 4000 bytes of ASCII code. Also note that one hexadecimal byte of address or data equals two bytes of ASCII.

Controls

The system controls the starting and stopping of the tape drive unit through the "remote" jack. When the remote is disabled (open), the tape drive motor is inoperative. In this mode the tape control buttons (play, rewind, record, etc.) have no effect. The UT62 monitor and the Editor/Assembler have a means of

enabling the remote jack to allow the user to rewind a tape. The **J command** prompts the user to rewind tapes and enables the remote line on both tape drive units. At this time the rewind button is active. When rewind is finished, the user simply *types any key* and the tapes are once again under system control. The user can now press the play or record buttons in preparation for a tape read or write operation.

10. Using the PROM Programmer

The PROM Programmer is a hardware and software package that facilitates user programming of RCA and other industry-standard PROM's. Specifically, the RCA CDP18U42, the Intel 2708, 2758, and 2716 UV-erasable PROM's, or equivalent PROM's can be programmed with this system. In addition, Intel 1702-type PROM's can be read (but not programmed) and copied into lower-power CDP18U42 CMOS PROM's or combined into larger-size PROM's. The PROM Programmer is included with the CDP-18S694V1 and V3 systems and may be added to the CDP18S693V1 and V3 systems by means of the PROM Programmer package CDP18S680.

The PROM Programmer package includes a plug-in Microboard module and a magnetic tape cassette containing the operating software. The software is designed for flexibility so that, in addition to the numerous basic operations provided, more sophisticated procedures can be derived from these building blocks. For example, a PROM can be copied from another PROM or programmed from a tape-resident file. The user can also combine two smaller PROM's into a larger one (two 2708's copied into one 2716). PROM's are automatically verified after programming. The user, however, can manually verify the erasure of a PROM, verify one PROM against another, or verify a PROM against a tape-resident file. These operations can be done with either positive (non-inverted) or negative (inverted) data.

An additional feature allows PROM's to be saved on a tape file (in a reloadable format) that can be used to generate masked ROM's. For PROM production purposes, a software feature facilitates rapid programming of many PROM's from the same source.

Installation

The PROM Programmer Module, as provided with the CDP18S694V1 and V3, is installed in the bottom slot of the chassis. When the module is to be used for programming PROM's, the user should remove the top cover and exchange the PROM Programmer module for the Microboard module in the top slot. By having the PROM programmer module in the top slot, access is provided to the three PROM programming sockets.

An external power supply is required as a programming voltage to be connected to jack J3, labelled PROG PWR. The voltage required for each PROM is given in the PROM manufacturer's data specification sheet. The supply must be well regulated and capable of supplying peak currents of 100 milliamperes. A -9-volt supply connected to J2 is also necessary when the programmer is used with a 1702 PROM. J1 is a ground connection. In addition to the external power supplies (Programming Power and -9 volts), voltages of +12 volts and -5 volts (both ±5%) must be supplied to the Microboard Universal Backplane pins 20 and 11, respectively, when 1702 or 2708 devices are to be operated on. Detailed specifications for the power supply requirements are given in Appendix I.

The PROM programmer module is assigned to Group Select 4. The module generates a signal on flag line EF1 of the CPU indicating whether the 1702/1842 switch is in the correct position.

Memory Organization and Use

The memory map for the PROM Programmer software is given in Fig. 17. The program loads starting at address 0000 and occupies the lowest two kilobytes of RAM. The hexadecimal addresses from 0800 to 0FFF comprise a data buffer area used to hold data moved to and from the PROM being operated on. The buffer memory is treated in pages, where a page is defined as 256 bytes. Thus, the first page of buffer memory occupies the hexadecimal addresses 0800 through 08FF and is called page 8; the second page, 0900 through 09FF, is called page 9; etc. The data of a 1702 or CDP18U42 occupy one page, a 2708 or 2758 four pages, and a 2716 eight pages.

The program asks for the starting page in the data buffer against which operations are to be performed.

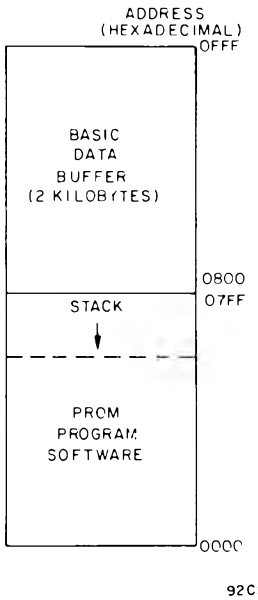


Fig. 17—Memory map for PROM Programmer software.

The response must be a one- or two-digit hexadecimal number equal to or greater than eight. Because a leading zero is assumed, only a single digit has to be entered for pages 8 through F. A number less than eight is not accepted by the program, thereby preventing accidental overwrite of some areas. Additional RAM can be used for additional page buffer areas.

The user can copy a PROM into any page (≥ 8), program a PROM from any page, or verify a PROM against any page. He does not necessarily have to start at page 8 and can take advantage of additional RAM if the MCDS is equipped with more than 4 kilobytes. However, when an object file is loaded into the buffer area, loading starts at address 0800. Likewise, when the buffer area is saved on a disk, the SAVEing begins at address 0800. Refer to the Section Operations and Commands for details on the operation of the object file reader and SAVE programs.

It is relatively easy to copy two smaller PROM's into one larger one. For example, to copy two 2758's into one 2716, the following steps would be followed.

(1) Copy the first 2758 starting at page 8. It will occupy two pages, addresses 0800 through 08FF.

(2) Copy the second 2758 starting at page A. It will occupy addresses 0G00 through 0FFF.

(3) Then, program a 2716 starting at page 8.

Reversing the procedure would permit programming several smaller PROM's from a larger one. Some care must be exercised in keeping track of where data is loaded in buffer memory for these combined operations. The software will not prevent overwriting of data in the buffer area because that may be the desired operation.

When programming takes place, data is taken from the starting page specified for the appropriate number of bytes. Thus, the data to be programmed must be contiguous in the buffer area.

Software Considerations

General Procedure

The PROM Programmer software is designed to prompt each step of a procedure so that operation of the system is self-explanatory and, it is hoped, relatively error-proof. For each type of operation (except SAVE) the user is asked to identify the type of PROM being used and whether positive (noninverted) or negative (inverted) data is to be used. When the program is started, and after completion of each operation, the following message is typed, asking what operation is next.

F, V, P, C, S, J, U?

The single letters refer to the following operations. The user should enter the appropriate letter.

F—Fill selected pages of the RAM buffer with 1's (positive logic) or 0's (negative logic)

V—Verify a PROM against the RAM buffer or a file

P—Program a PROM from the RAM buffer or a file

C—Copy a PROM into the RAM buffer

S—Save the RAM buffer on a file

J—Rewind the tape

U—Exit to the monitor

For all except the SAVE operation, the PROM Programmer asks for the PROM type by the message

TYPE#?

The user should enter one of the following:

1702, 1842, 2708, 2758, or 2716

followed by a carriage return (CR).

The next question asked is whether the operation is to be performed with positive logic (non-inverted data) or negative logic (inverted data).

LOGIC =P, N?

The user replies by typing a P for positive logic or an N for negative logic.

These three prompts are common for the V, C, P, and F commands. All operations also have in common the manipulation of data in the RAM buffer area previously described. Data to be programmed goes to a PROM from the buffer area. Likewise, data copied from a PROM goes into the buffer, etc. Succeeding questions ferret out the area of RAM to be used.

When an operation is completed, the program types

DONE

In the case of a verify (V) or programming (D) sequence, it asks if the user wants to repeat the operation with a question such as

REPROGRAM 1842?

The user responds with a "Y" for yes or an "N" for no. Because a "yes" response starts another programming

cycle immediately, the new PROM should be in place before an affirmative response is given. A "no" response restarts the program.

The Section **Operations and Commands** gives details and examples of each operation.

Error Messages

Whenever the program receives an inappropriate response to a question, it types a "?" and repeats the question. It continues this reaction to each incorrect response until an acceptable one is received. Then, it will proceed.

The "?" is also generated if a number less than 8 is entered in response to the PAGE? question or if the user asks to program a 1702 (a non-existent function).

The following error messages may also be generated when a tape-read operation is being performed:

NO OBJECT CODE

or

FORMAT ERROR

or

LOAD ABORTED

The last message follows a non-recoverable error message.

Aborting Operations

When a PROM is being verified, a list of errors is generated showing any mismatches between the PROM data and what is in the RAM buffer. A long typeout can be halted by pressing the BREAK key. Because the BREAK key is sampled at the end of each typed line, it must be held down long enough to be "seen" by the program. The message

*****ABORTED*****

is typed and, when the BREAK key is released, the program restarts from the beginning.

A programming operation can be aborted by pressing and holding the BREAK key. Again, the message is typed and the program restarts. Response to this key may take a few seconds.

Correcting Typing Errors

All numbers entered in response to a question must be followed by a carriage return (CR). The program looks only at the last two or four digits, as appropriate, that have been entered before the CR. Any preceding digits are ignored. This feature provides a convenient means of correcting typing errors. The user needs only to continue typing until the correct digits have been entered before pushing CR. For example, typing

27142716(CR) is equivalent to

2716(CR)

Or, in response to the PAGE# = question, typing

41008(CR) is equivalent to

08(CR)

Inappropriate responses result in a "?" appearing and the question reasked as described in the subsection "Error Messages". These kinds of errors are corrected by simply entering a valid response on the next try.

Errors of the third kind, in which an incorrect though valid response is made, must be corrected by pressing the RESET button on the MCDS and restarting the program with the monitor program's P command. For example, if a "V" command is inadvertently entered instead of the "P" command, the program should be restarted to correct the situation.

Precautions

A PROM should never be inserted or removed with power on. The user should always make sure that the power ON/OFF switch S2 on the PROM programmer module is in the OFF position before inserting or removing a PROM.

To prevent any potential problems, the external power supplies (programming power and—9 volts), when present, should be sequenced on simultaneously or after the MCDS is turned on and shut off before or at the same time the MCDS is shut off.

There is another switch (S1) on the PROM Programmer board that selects between a 1702 and CDP18U42 configuration for socket 1.

NOTE: This switch should never be in the 1702 position if a CDP18U42 is in the socket and the —9-volt supply active. If such were the case, —9 volts would be connected to an input pin of the CDP18U42, and a destructive latch-up condition could be triggered in the device. If the program generates the warning message

SET SWITCH!

it detects the wrong position after the program has started and waits before proceeding. This warning, however, should not be relied on to prevent damage to the device. **Before inserting a CDP18U42, be sure switch S1 is in the correct position.**

The user should be sure to insert each PROM into its correct socket and be sure that it is properly oriented. Pin 1 is the pin adjacent to the handle of the zero-insertion force sockets. **Never have more than one PROM in the module at any time.** Because the PROM sockets share common address and data buses, contention or loading problems could be caused if more than one device is inserted. Additionally, extraneous signals to a socket which should be empty might cause harm to a device left in it.

The PROM programmer software derives its timing from the 2.0-MHz clock frequency of the crystal on the CDP18S601 Microboard Computer Module. If this frequency is changed, the results are unpredictable.

Also, note the power requirements for the PROM Programmer module as given in Appendix I. If additional modules are installed in the MCDS that could tax the internal power supplies to their limits, these modules should be removed before attempting to run the programmer.

Before starting a programming operation, be sure that the external programming power supply (connected to J3) is at the proper voltage. Excessive voltage can cause permanent damage to a PROM.

It is recommended that the programmer module not be run on an extender card in the MCDS system. An extender card will exaggerate bus-loading effects and may lead to intermittent operation of the programmer, particularly in systems already loaded with extra I/O or memory modules. If anomalous operation is observed, a first step should be to remove any extender cards and extraneous modules from the system.

Operations and Commands

Following is a discussion of each command and operating sequence.

Programming Operations— The "P" Command

The system is capable of programming the RCA CDP18U42 CMOS PROM and Intel 2708, 2758, and 2716 type PROM's. Data to be programmed comes from the RAM Buffer area, pages 8 and up in the MCDS RAM. The data must first be loaded into the appropriate page(s) by COPYing another PROM into them or by loading the appropriate section of a tape file into them. The file loading, when requested, leads automatically into a programming cycle. Refer to the section, **File Reader Subroutines** for a detailed discussion of the file loaders. Programming begins immediately from the specified page.

Procedure

The program always begins by printing

MCDS PROM PROGRAMMER VER X

F, V, P, C, S, J, U?

After the user types the letter 'P", the program asks

TYPE #?

The user should enter one of the following:

1842, 2708, 2758, or 2716

followed by a (CR).

The program next asks if operation is to be with true or inverted data by typing

LOGIC=P, N?

A "P" response causes the operation to occur with true (non-inverted) data; an "N" response causes the operation to proceed using an inverse of the data. For example, to copy one PROM into another, inverting data, one could copy the master PROM using positive logic and program the copy PROM using negative logic or vice-versa. Negative logic would not be used for both the copy and program step because that would give a double inversion.

The program next asks

(T)APE OR (R)AM?

The user should enter the letter "R" if the data to be programmed is already in the buffer RAM.

The next prompt is

PAGE# =

asking for the first page on which data is located. Data is taken from that page and subsequent pages as required for the PROM being programmed.

If the user enters the letter "T", indicating data to be programmed is on a tape file, the program will ask for the unit by typing

TAPE#

The user should respond with either a "0" or a "1".

Next, the program asks for the address range of the PROM by typing

LOWEST PROM ADDRESS (XX00)?

The user should respond with a four-digit number followed by a (CR). This number should be the lowest address of data to be programmed into the PROM. For instance, if a 2704 PROM is to be programmed for data addresses 0200 through 03FF, then 0200 would be typed as a response. A default value of 0000 is entered if the user hits the carriage return without typing in a number. If a number is entered, **all four digits must be entered.** The file reader searches the object file and loads only the data falling between the lowest address specified and up to 2 kilobytes above that address. The appropriate data is copied into RAM starting at page 8 and then the actual programming begins automatically.

Note that programming will be initiated when the 2-kilobyte RAM area is filled or an end-of-file marker is read from tape. Thus, if the user enters an improper start address, the PROM will be programmed with invalid data.

Any programming operation is automatically followed by a VERIFY sequence. If the verify is successful, the program types

DONE

If programming is not successful, a list of errors is typed. This list can be stopped by depressing the BREAK key until the

*****ABORTED*****

message is typed.

If programming has been done successfully, a message such as

REPROGRAM 1842?

appears, identifying the PROM type just programmed. If the user does not want to program another device of the same type with the same information, he should type "N" for NO. If the programming of another PROM of the same type is required, the user should take the following steps:

—Shut off power switch S2
—Remove the PROM just programmed
—Insert the new PROM to be programmed
—Turn on switch S2
—Type the letter "Y" for yes on the terminal.

Another programming sequence will begin immediately.

During a programming operation, the LED indicator labeled PROG on the programmer board is ON, providing a visual indication of operation. The programming operation can be halted at any time by

pressing and holding the BREAK key until the message

ABORTED

is typed.

The programming time required for each PROM type is given in Table XVI.

Table XVI Programming Times—Typical Values.

Type No.	Time
CDP18U42	3 seconds
2708	2 minutes 45 seconds
2716	1 minute 45 seconds
2758	50 seconds

Examples

In the following examples, system-generated output is underlined.

1. Program a 2758 from RAM pages 8, 9

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? P
TYPE #? 2758
LOGIC=P, N? P
(T)APE OR (R)AM? R      PAGE# = 8
DONE
REPROGRAM 2758? N
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U?
```

2. Program two CDP1842's with the same data from a tape on drive 1. Program for address 0100 to 01FF.

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? P
TYPE #? 1842
LOGIC=P, N? P
(T)APE OR (R)AM? T
TAPE#? 1
LOWEST PROM ADDRESS (XX00)? 0100
LOADING COMPLETED    ..FILE LOADED
                     ..PROGRAMMING BEGINS
                     ..PROGRAMMING AND
                     ..VERIFY DONE
REPROGRAM 1842? Y    ..INSERT NEXT 1842,
                     ..THEN TYPE Y.
DONE                 ..PROGRAMMING AND
                     ..VERIFY DONE
REPROGRAM 1842? N    ..RESTART PROGRAM
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? U    ..RETURN TO MONITOR
*
```

Verify Operations—The "V" Command

Verify operations compare a selected area of the RAM buffer against the contents of a PROM. As with the programming operation, the buffer is loaded by either COPYing a PROM into it or loading data from a file. The user can select the page(s) against which a PROM is to be verified, or, if he chooses to use a file, the appropriate data is loaded starting at page 8 and automatically verified from there. See subsection "File Reader Subroutines" later in this section for details on the file reader operation.

Procedure

A verify operation is automatically done after a PROGRAM or COPY operation. If there are no errors, the DONE message appears. Any mismatch between a PROM address and the corresponding buffer address results in an error listing such as

ERR;M(0800)=FF, PROM=00

This error listing says that location 0800 in the RAM buffer contains an FF, but the corresponding location of the PROM (which would be its first address in this case) contains a 00. A long error listing can be stopped by pressing and holding the BREAK key until the message

ABORTED

is typed. The program then restarts from the beginning when the BREAK key is released.

The program begins by asking for the type of operation to be performed, for the PROM type number, and whether the operation is with positive or negative logic. The verify command has exactly the same command format and requirements as the "P" command.

At the end of a verify operation, the program will ask if another PROM of the same type is to be verified against the same RAM data. If the answer is no, the user should type an "N". If the answer is yes, the following steps should be taken.

—Shut off power switch S2
—Remove the PROM just verified
—Insert the new PROM to be verified
—Turn on switch S2
—Type the letter Y for yes on the terminal

Examples

1. Verify a 2708 against RAM pages 9 through C.

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? V
TYPE #? 2716
LOGIC=P, N? P
(T)APE OR (R)AM? R      PAGE# =9
DONE
REVERIFY 2708? N
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U?
```

2. Verify a 2716 against a tape file. Verify against addresses 0500 through 0CFF of the file.

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? V
TYPE #? 2716
LOGIC=P, N? P
(T)APE OR (R)AM? T
TAPE#? 1
```

```
LOWEST PROM ADDRESS (XX00)? 0500
LOADING COMPLETED
ERR:M(0800)=18, PROM=FF ..ERROR LSTG STARTS
ERR:M(0801)=C0, PROM=FF
ERR:M(0802)=04, PROM=FF ..BREAK KEY PRESSED
***ABORTED***
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U?
```

Copy Operations— The "C" Command

Procedure

The copy operation reads the contents of a PROM into the selected page(s) of the buffer RAM. Make sure there is sufficient free RAM to hold all the data to be copied. For example, copying a 2716 requires two kilobytes of RAM buffer and the copy must begin on page 8 if there are only four kilobytes of RAM in a MCDS system. The program does not test for the presence of RAM at the assigned page(s). A copy operation is automatically followed by a verify. Upon completion of a copying operation, the program responds with a "DONE" message or a list of errors if the copy did not verify. Errors could be due to a hardware problem or attempting to copy into non-existent RAM.

A direct copy of the PROM is made if positive logic is chosen; the data is inverted if negative logic is chosen. A COPY followed by a PROGRAM operation is the sequence used to reproduce PROM's. Refer to the next Section for examples of compound operations.

Example

Copy a 2758 into RAM pages 8, 9, A and B.

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? C
TYPE #? 2758
LOGIC=P, N? P      PAGE# =8
DONE
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U?
```

The Fill Command

Procedure

This command is used to fill selected page(s) of the buffer RAM with all 1's for positive logic, or all 0's for negative logic. The user should respond to the PAGE#= prompt with the starting page, as usual. The number of pages that will be filled is determined by the PROM type number entered—1 page for 1702 or CDP18U42, 4 pages for a 2758, etc.

This command is used in verifying that a PROM has been completely erased. The erased PROM is VERIFY'ed against a buffer filled with 1's or 0's, as appropriate.

Example

Fill RAM pages B through E with 0's (4 pages).

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? F
TYPE #? 2708      ..2708 IS 4-PG DEVICE
LOGIC=P, N? N     PAGE# =B
DONE
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U?
```

The "U" Command

Procedure

The "U" command returns control to the monitor program which responds by printing an asterisk (*) prompt.

Example

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? U
*
```

The "J" Command

Procedure

The "J" command allows the user to rewind tapes.

Example

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? J
REWIND TAPES, THEN TYPE ANY KEY X
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U?
```

The SAVE Operation— The "S" Command

The SAVE routine allows the user to copy any number of pages of the buffer RAM onto a file in reloadable format. The file thus generated will be in ASCII-Hex and can be used to program other PROM's. Also, it can be loaded into RAM for a ROM emulator function (a Micromonitor feature) or it can serve as the data for generation of masked ROM's such as the CDP1831/33.

Procedure

The prompts for this command are different from the previous ones. The user is not asked for the type of PROM or the starting page. Saving always starts on page 8 and is done in positive logic. The number of pages to be saved is designated in hexadecimal—from 1 to F7. The S command, in conjunction with the tape file reader and other commands, can be used to merge or truncate files and combine several smaller PROM's for the generation of one larger ROM. See the next section for examples of compound command usage.

When the SAVE operation is started, it asks for the number of pages of data to be saved on tape by typing

```
# PAGES?
```

The SAVE operation then asks for the tape number the data is to be written to by typing

```
TAPE#?
```

The user should respond with either a "0" or a "1". The response to the prompt

LOWEST PROM ADDRESS (XX00)?

is used to generate the starting address on the file. For example, the response "0100" causes data to be written on the file as follows:

```
I
0100 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX;
0110 XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX;
                    etc.
```

Be sure the correct starting address is assigned.

SAVE'd files on tape cassette will have a leader and trailer of nulls appended automatically.

Example

Save RAM pages 08 through 12 (hex) under the file on a tape in drive #1. The data is from a PROM with starting address 0200.

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? S
# PAGES? A ..TEN PAGES WILL BE SAVED
TAPE#? 1
LOWEST PROM ADDRESS (XX00)? 0200
DONE
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U?
```

File Reader Subroutines

The File Reader Subroutine is a progam called by either the PROGRAM or VERIFY routines. It loads data from a file into the buffer RAM starting at page 8. It is an intelligent loader that will search through a file and load data in the address range of interest (+2 kilobytes from the initial address) ignoring extraneous data. It will handle either hex or listing files created by the COSMAC assemblers and files created by the monitor program (W command). When the user replies to the prompt

LOWEST PROM ADDRESS (XX00)?

the program takes that address as the first address in the range and adds 2K to get the last address. It will thus load up to 2 kilobytes of data, filling the basic buffer (but not writing into any additional CDS RAM) if there is sufficient data. Of course, if the file contains less than 2 kilobytes in the address range, some of the buffer will not be used. Once a 2-kilobyte file is loaded, small PROM's can be programmed from successive pages of the buffer. Refer to the next section for an example.

If no data is found, the program types

NO OBJECT CODE

In the tape cassette system, the program will restart from the beginning.

Format errors detected on the cassette will cause the program to type the following messages and restart from the beginning.

FORMAT ERROR
LOAD ABORTED

When loading starts, the message

LOADING

is typed. At the completion of a successful load, the word COMPLETED is added to the sentence. The program then goes automatically into the program or verify operation.

NOTE: In response to the prompt

LOWEST PROM ADDRESS (XX00)?

a full four-digit number followed by a carriage return (CR) should be entered. If the user hits CR without entering any numbers, the default address of 0000 is used. Typing mistakes can be corrected by continuing to type until the correct four digits have been entered before hitting CR. Only the last four digits before the CR are recognized and only the first two of the group are meaningful—the last two are always taken as 00. Below are some examples.

Response	Yields
0200(CR)	0200
0251(CR)	0200
2(CR)	0000
030244(CR)	0200
03029(CR)	3000
(CR)	0000

Application Hints

It is good practice to verify that a PROM is erased before attempting to program it. A suitable technique is to use the "F" command to fill the buffer with 1's or 0's and then the "V" command to verify the PROM against it. When a PROM is being programmed to work in an RCA 1800-series microprocessor-based system, it is advisable to first fill the buffer with all 0's before loading in the data to be programmed. This step will set data in the unused addresses of the PROM to 00, which is an IDLE instruction in the RCA 1800-series instruction set. Thus, an erratic program fetching from an erroneous address will pick up the idle instruction and stop. A PROM thus programmed must use the same procedure when it is verified. That is, the buffer is first filled with 0's, the data loaded, and the verify operation is made. This technique assures that no "errors" will appear when the programmed "00" locations are verified against random data left in the buffer.

An application note (ICAN-6847) describing a hardware adjunct that can be used to program Intel 2732 PROM's with this system is available on request.

Examples of Compound Operations

This section gives five examples of compound operations.

Example 1

Program a CDP18U42 from a 1702.

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? C      ..COPY THE 1702 INTO RAM
TYPE #? 1702
LOGIC=P, N,? P      PAGE#=8
DONE      ..REMOVE 1702, SET SWITCH,
          ..INSERT 1842
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? P
TYPE #? 1842
LOGIC=P, N? P
(T)APE OR (R)AM? R      PAGE#=8
DONE
REPROGRAM 1842? N      ..RESTART PROGRAM
```

Example 2

Verify that a 2758 has been erased, then program it from a file. The PROM will occupy addresses from 400-5FF.

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? F
TYPE#? 2758
LOGIC=P, N? P      PAGE#=8
    ..FILL PGS. 8, 9, A, AND B WITH FF'S
DONE
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? V
TYPE #? 2758
LOGIC=P, N? P
(T)APE OR (R)AM? R      PAGE#=8
DONE      ..ERASURE VERIFIED—NOW PROGRAM
REVERIFY 2758? N
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? P
TYPE#? 2758
LOGIC=P, N? P
(T)APE OR (R)AM? T
TAPE#? 1
LOWEST PROM ADDRESS (XX00)? 0400
LOADING COMPLETED
DONE      ..PROGRAMMED AND VERIFIED
REPROGRAM 2758? N      ..RESTART PROGRAM
```

Example 3

Combine and save two CDP18U42's on tape to make a CDP1831 ROM with address range 0100-02FF.

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? C      ..COPY 1ST PROM INTO PG 8
TYPE #? 1842
LOGIC=P, N? P      PAGE# =8
DONE
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? C      ..COPY 2ND PROM INTO PG 9
TYPE #? 1842
LOGIC=P, N? P      PAGE# =9
DONE
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? S
# PAGES? 2
TAPE#? 1
LOWEST PROM ADDRESS (XX00)? 0100
DONE
```

Example 4

Program two 2708's from a file. One occupies addresses 0000-03FF and the other 0400-07FF.

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? P
TYPE #? 2708
LOGIC=P, N? P
(T)APE OR (R)AM? T
TAPE#? 0
LOWEST PROM ADDRESS (XX00)? (CR)
LOADING COMPLETED
DONE      ..1ST PROM PROGRAMMED
          ..FOR 0000-03FF
REPROGRAM 2708? N
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? P
TYPE#? 2708
LOGIC=P, N? P
(T)APE OR (R)AM? R      PAGE#= C
DONE      ..FOR 0400-07FF
REPROGRAM 2708? N
```

Example 5

Program two 2716's from a file. One occupies addresses 0000-07FF and the other 0800-0FFF.

```
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? P
TYPE #? 2716
LOGIC=P, N? P
(T)APE OR (R)AM? T
TAPE#? 0      ..LOAD 1ST ADDRESS RANGE
LOWEST PROM ADDRESS (XX00)? (CR)
LOADING COMPLETED
DONE      ..1ST PROM PROGRAMMED
REPROGRAM 2716? N
MCDS PROM PROGRAMMER VER X
F, V, P, C, S, J, U? P
TYPE #? 2716
LOGIC=P, N? P
(T)APE OR (R)AM? T
TAPE#? 0      ..LOAD 2ND ADDRESS RANGE
LOWEST PROM ADDRESS (XX00)? 0800
LOADING COMPLETED
DONE      ..2ND PROM PROGRAMMED
REPROGRAM 2716? N
```

Appendix A—
Specifications for the Microboard Computer Development System (MCDS) CDP18S693 and CDP18S694

CDP18S693V1 and CDP18S693V3 System Components

CDP18S601 Microboard Computer
CDP18S652 Combination Memory and Tape I/O Control Module
8-Card Chassis with Protective Base and Cover
CDP18S023V1 or CDP18S023V3 Power Converter*
CDP18S516 EIA Terminal Interface Cable (RS232C)
UT62 ROM-based Monitor Program
CDP1841 BASIC3; ROM-based Extended Basic Program
CDP18S810 Audio Cassette Recorder Unit
Cassette Interface Cable
Technical Literature

CDP18S694V1 and CDP18S694V3 System Components

All the components of the CDP18S693 system plus:
CDP18S810 Audio Cassette Recorder Unit
Cassette Interface Cable
PROM Programmer Module
Cassette Tape with PROM Programmer Software
Text Editor, ROM-based
Level I Assembler, ROM-based

Memory, I/O, and Control Specifications

RAM

4 kilobytes on CDP18S601 at 0000H—0FFFH
1 kilobyte on CDP18S652 at 8C00H—8FFFH

ROM

4 sockets for up to 8 kilobytes on CDP18S601
2 kilobytes preprogrammed with UT62 on CDP-18S652
12 kilobytes preprogrammed with BASIC3 on CDP18S652
6 kilobytes preprogrammed with Editor/Assembler on CDP18S652

Parallel I/O

20 lines, programmable
4 external flag inputs
1 Q line output

Serial I/O

RS232C or 20-mA loop, software driven, automatic baud rate selection of 110, 300, or 1200
Two audio cassette tape unit channels with start/stop controls

System Control

RESET/RUN switch linkable to start running at 8000H for UT62 or at 0000H for user program

*Because the MCDS operates from a wall-type 5-volt power supply, care must be taken when additional Microboards are used with the system. As a general precaution, it is recommended that a separate power supply be used for any additions to the system.

Appendix B—
UT62 Monitor Program Listing

```
!M
0000 ;              0001  ..***********************************************
0000 ;              0002 ..NAME:  UT62/4
0000 ;              0003 ..DESC:  RCA MICROBOARD MONITOR FOR CASSETTE I/O
0000 ;              0004 ..DATE:  6/16/82
0000 ;              0005
0000 ;              0006
0000 ;              0007 ..       UT62 IS A MONITOR PROGRAM DESIGNED TO
0000 ;              0008 ..       EXAMINE MEMORY, ALTER MEMORY, AND TO
0000 ;              0009 ..       TO BEGIN PROGRAM EXECUTION AT A GIVEN
0000 ;              0010 ..       LOCATION. ALSO, PROGRAMS CAN BE SAVED ON TAPE. THESE FUNCTIONS
0000 ;              0011 ..       ARE ACCOMPLISHED THROUGH A SERIES OF MONITOR
0000 ;              0012 ..       COMMANDS WHICH
0000 ;              0013 ..       ARE INITIATED BY TYPING A,B,C,D,E,F,I,J,M,P,R,S,OR W.
0000 ;              0014 ..       THE  FUNCTIONS INCLUDE MEMORY INSERT[I],
0000 ;              0015 ..       MEMORY DISPLAY[D], MEMORY MOVE[M],
0000 ;              0016 ..       MEMORY FILL[F], MEMORY SUBSTITUTE[S],READ TAPE[R],
0000 ;              0017 ..       RUN EDITOR[E],RUN ASSEMBLER[A],RUN BASIC[B],
0000 ;              0018 ..       WRITE TAPE[W],REWIND TAPES[J],COPY TAPE TO SCREEN [C],
0000 ;              0019 ..       AND RUN PROGRAM[P].
0000 ;              0020 ..       TAPE READ LOADS MEMORY FROM TAPE.
0000 ;              0021 ..       TAPE WRITE SAVES MEMORY ON TAPE FROM START TO
0000 ;              0022 ..       END ADDRESS AS SPECIFIED.
0000 ;              0023 ..       ALSO INCLUDED ARE THE STANDARD READ AND
0000 ;              0024 ..       TYPE ROUTINES WHICH PROVIDE COMMUNICATION
0000 ;              0025 ..       WITH THE USERS TERMINAL.
0000 ;              0026 ..       UPON INVOCATION, THE CONTENTS OF THE
0000 ;              0027 ..       CPU REGISTERS ARE SAVED IN RAM AT #8C00.
0000 ;              0028 ..       THE CONTENTS OF R0, R1, AND R4.1 ARE
0000 ;              0029 ..       DESTROYED, HOWEVER, BY THE PROCESS.
0000 ;              0030 ..       THEIR CONTENTS CAN BE EXAMINED BY DOING
0000 ;              0031 ..       D8C00 FOR 20 BYTES.
0000 ;              0032
0000 ;              0033 ..***********************************************
0000 ;              0034 ..       SYSTEM EQUATES, CONSTANTS & OFFSETS
0000 ;              0035 ..***********************************************
0000 ;              0036
0000 ;              0037 ..      REGISTER ASSIGNMENTS - GEN. & UTILITIES
0000 ;              0038
0000 ;              0039 SP=#02..STACKPOINTER
0000 ;              0040 PC=#03..PROGRAM COUNTER
0000 ;              0041 CALL=#04..CALL ROUTINE COUNTER
0000 ;              0042 RETN=#05..RETURN ROUTINE COUNTER
0000 ;              0043 LINK=#06..SUBROUTINE DATA LINK
0000 ;              0044 BYTSTR=#07..TEMPORARY STORAGE DURING WRITE
0000 ;              0045 SRTADD=#07..START ADDRESS FOR TAPE WRITE
0000 ;              0046 PARITY=#07..FOR TAPE ROUTINES
0000 ;              0047 TMPRG1=#07...TEMPORARY REGISTER
0000 ;              0048 BITRDR=#08...POINTS TO THE ROUTINE TO READ TAPED BITS
0000 ;              0049 BITWRT=#08...POINTS TO ROUTINE TO WRITE A BIT TO TAPE
0000 ;              0050 ENDADD=#08...HOLDS END ADDRESS FOR WRITE TO TAPE
0000 ;              0051 ADRPTR=#08...HOLDS ADDRESS DURING STORE FROM TP
0000 ;              0052 TMPRG2=#08...TEMPORARY REGISTER
0000 ;              0053 LDRCNT=#09...COUNTER FOR TAPE LEADER
0000 ;              0054 BITCNT=#09...COUNTER FOR BITS PER WORD FROM TAPE
0000 ;              0055 TMPRG3=#09...TEMPORARY REGISTER
0000 ;              0056 BUFPTR=#0A...POINTS TO TAPE READ/WRITE BUFFERS
0000 ;              0057 CNT=#0A...BYTE COUNT
0000 ;              0058 SRC=#0B...SOURCE REGISTER
0000 ;              0059 TPTR=#0B...TABLE POINTER
```

```
0000 ;              0060 BLKFLG=#0B...FLAG TO SIGNAL TO WRITE A BLOCK
0000 ;              0061 DELAY=#0C...DELAY ROUTINE COUNTER
0000 ;              0062 PTER=#0C...IOCB PTR
0000 ;              0063 PTR=#0C...DITTO
0000 ;              0064 ASL=#0D...HEX INPUT REGISTER
0000 ;              0065 DEST=#0D...DESTINATION REGISTER
0000 ;              0066 AUX=#0E...AUX.1 HOLDS BIT TIME CONSTANT
0000 ;              0067 CHAR=#0F...STORES ASCII I/O
0000 ;              0068 FLAG=#0F...FLAGS FOR TAPE
0000 ;              0069
0000 ;              0070 ..      RAM / ROM ALLOCATIONS
0000 ;              0071
0000 ;              0072 RDBUFF=#8E7F
0000 ;              0073 TOPSTK=#8CFF
0000 ;              0074 WRAM=#8C1F
0000 ;              0075 WRTBUF=#8E80
0000 ;              0076 UT62=#8000
0000 ;              0077
0000 ;              0078 ..      ROM BASED PROGRAM HIGH ORDER ADDRESS VALUES
0000 ;              0079
0000 ;              0080 ABAS=#B0
0000 ;              0081 AEDIT=#90
0000 ;              0082 AASM=#9B
0000 ;              0083 PRMPGR=#E0
0000 ;              0084
0000 ;              0085 ..      ASCII CONTROL CHARACTERS
0000 ;              0086
0000 ;              0087 NULL=#00        ...NULL
0000 ;              0088 COMMA=#2C       ...COMMA
0000 ;              0089 SEMCOL=#3B      ...SEMICOLON
0000 ;              0090 BS=#08          ...BACK SPACE
0000 ;              0091 LF=#0A          ...LINE FEED
0000 ;              0092 CR=#0D          ...CARRIAGE RETURN
0000 ;              0093 EOF=#13         ...END OF FILE
0000 ;              0094 SPACE=#20       ...SPACE (WORD SEPARATOR)
0000 ;              0095 CRLF=#0D0A      ...RETURN,LINEFEED
0000 ;              0096
0000 ;              0097 ..      CONSTANTS
0000 ;              0098
0000 ;              0099 BDSEL=#01...PORT FOR TWO LEVEL I/O SELECT
0000 ;              0100 BTHDRV=#C0 ..SELECTS BOTH TAPE DRIVES
0000 ;              0101 DRIVE0=#40...SELECTS TAPE DRIVE 0
0000 ;              0102 DRIVE1=#80...SELECTS TAPE DRIVE 1
0000 ;              0103 HIGH=#01...WRITE A HIGH TO TAPE
0000 ;              0104 LNECNT=#0F...# OF BYTES PER LINE IN DISPLAY ROUTINE
0000 ;              0105 LINES=#14...NUMBER OF LINES PER SCREEN LOAD
0000 ;              0106 LOW=#00...WRITE A LOW TO TAPE
0000 ;              0107 PGMSRT=#0005 ..START ADDRESS FOR INIT1
0000 ;              0108 PROMPT=#2A...PROMPT CHARACTER
0000 ;              0109 TAPEIO=#02...SELECTS TAPE RECORDER
0000 ;              0110 TPEOFF=#00...TURN OFF TAPES
0000 ;              0111 TPESEL=#04...PORT TO SELECT TAPES
0000 ;              0112 TPEWRT=#05...PORT TO WRITE TO TAPE
0000 ;              0113 TRMINL=#01...SELECTS THE TERMINAL
0000 ;              0114
0000 ;              0115 ..*************************************************************
0000 ;              0116                ORG UT62
8000 ;              0117
8000 7100;          0118        DIS;,0              ..DISABLE
8002 F880B0;        0119        LDI A.1(UT62) ;PHI R0..ESTABLISH PROGRAM COUNTER AT
```

```
8005 ;              0120                          ..8000 HEX
8005 ;              0121
8005 ;              0122 ..*****************************************************
8005 ;              0123 ..                 REGISTER SAVE
8005 ;              0124 ..       SAVES CONTENTS OF THE CPU REGISTERS @#8C00.
8005 ;              0125 ..*****************************************************
8005 ;              0126
8005 F88CB1;        0127         LDI A.1(WRAM-1) ;PHI R1 ..THIS SEQUENCE PERFORMS
8008 F81EA1;        0128         LDI A.0(WRAM-1) ;PLO R1..THE REGISTER SAVE
800B F8A0B4;        0129         LDI #A0;PHI R4   ..MODIFIED INSTRUCTION
800E E1;            0130         SEX R1
800F ;              0131                          ..SET UP SEP INSTRUCTION
800F F8D051;        0132 LOOP:   LDI #D0;STR R1    ..FOR RETURN
8012 F3;            0133         XOR             ..CHECK IT WROTE
8013 3A29;          0134         BNZ UT62A
8015 21;            0135         DEC R1            ..PREPARE FOR 9N INST.
8016 94FC70;        0136         GHI R4;ADI #70
8019 331D;          0137         BDF *+4
801B FC21;          0138         ADI #21        ..NO, 8N  ;9N
801D FC7F;          0139         ADI #7F          ..YES, 9N  ;8(N-1)
801F B4;            0140         PHI R4
8020 51;            0141         STR R1             ..SAVE INST. & EXECUTE IT
8021 D1;            0142         SEP R1
8022 73;            0143         STXD         ..STORE RESULT IN RAM
8023 21;            0144         DEC R1             ..BACK UP & CHECK IF
8024 94FB91;        0145         GHI R4;XRI #91   ..STORAGE DONE?
8027 3A0F;          0146         BNZ LOOP     ..NEXT BYTE
8029 ;              0147
8029 C08381;        0148 UT62A:  LBR INIT
802C ;              0149
802C D480FE;        0150 START:  SEP CALL;,A(TIMALC)
802F ;              0151
802F ;              0152 ..*****************************************************************
802F ;              0153 ..                 OUTPUT THE UTILITY PROMPT
802F ;              0154 ..*****************************************************************
802F ;              0155
802F D483FF;        0156 PRMPT:  SEP CALL;,A(TPOFF)       ..TURN OFF TAPES,SEL RCA GRP
8032 D483F0;        0157         SEP CALL;,A(OSTRNG)       ..OUTPUT PROMPT TO USER TERMINAL
8035 0D0A2A00;      0158         ,A(CRLF),PROMPT,0
8039 ;              0159
8039 ;              0160
8039 ;              0161 ..*****************************************************************
8039 ;              0162 ..              MONITOR COMMAND INTERPRETER
8039 ;              0163 ..       FETCHES THE ADDRESS FROM THE COMMAND TABLE AND SETS
8039 ;              0164 ..       THE PROGRAM COUNTER TO IT
8039 ;              0165 ..    REG USED:  PTR, CHAR, SP, ASL
8039 ;              0166 ..*****************************************************************
8039 F85AAB;        0167         LDI A.0(TAB2);PLO TPTR
803C F880BB;        0168         LDI A.1(TAB2);PHI TPTR   ..GET COMMAND TABLE ADDR
803F ;              0169
803F D4813E;        0170 SCNLTR: SEP CALL;,A(READ)  ..READ COMMAND
8042 EB;            0171         SEX TPTR
8043 ;              0172
8043 9FF3;          0173 SCAN:   GHI CHAR;XOR       ..LOOK FOR MATCH
8045 1B;            0174         INC TPTR
8046 324F;          0175         BZ GOTHRE ..YES, COMPUTE ADDRESS
8048 1B1B;          0176         INC TPTR;INC TPTR  ..INC THE POINTER TO THE
804A 0B;            0177         LDN TPTR              ..NEXT COMMAND
804B 3285;          0178         BZ ERROR  ..ERROR IF END OF TABLE
804D 3043;          0179         BR SCAN          ..ELSE CONTINUE LOOKING
```

```
804F ;              0180
804F 2222;          0181 GOTHRE: DEC SP; DEC SP     ..FAKE IT FOR THE RETURN
8051 ;              0182                            ..PICK UP COMMAND ADDRESS
8051 4BB6;          0183         LDA TPTR;PHI LINK  ..AND TRANSFER TO THE
8053 ;              0184                            ..SUBROUTINE BY EXECUTING
8053 4BA6;          0185         LDA TPTR;PLO LINK  ..A RETURN INSTRUCTION
8055 F800BDAD;      0186         LDI 0 ;PHI ASL;PLO ASL
8059 D5;            0187         SEP R5               ..P=3,X=2,R4 ;SEP CALL;,,R5 ;RETURN,R2=#8CFF
805A ;              0188
805A ;              0189 ..*****************************************************************
805A ;              0190 ..                    COMMAND TABLES
805A ;              0191 ..*****************************************************************
805A ;              0192
805A 4482BD;        0193 TAB2:    ,T'D',A(DISPLY)     ..MEMORY DISPLAY
805D 4983A7;        0194          ,T'I',A(INSERT)      ..INSERT INTO MEMORY
8060 4D82F7;        0195          ,T'M',A(MOVE)        ..MOVE A BLOCK OF MEMORY
8063 468240;        0196          ,T'F',A(FILL)        ..FILL A BLOCK OF MEMORY
8066 538099;        0197          ,T'S',A(SUBST)       ..BYTE SUBSTITUTION
8069 508534;        0198          ,T'P',A(RUN)         ..RUN A USER PROGRAM
806C 528474;        0199          ,T'R',A(TPLOAD)      ..READ FROM TAPE TO MEMORY
806F 4A852E;        0200          ,T'J',A(TPWIND)      ..REWIND TAPES
8072 5786D4;        0201          ,T'W',A(TPWRIT)      ..WRITE ONTO TAPE FROM MEMORY
8075 4585E7;        0202          ,T'E',A(EDITOR)      ..START THE EDITOR
8078 4185EC;        0203          ,T'A',A(ASSEMB)      ..START THE ASSEMBLER
807B 4285F1;        0204          ,T'B',A(BASIC)       ..START BASIC
807E 5A85F6;        0205          ,T'Z',A(PROMIT)      ..START CPU AT #E000,P&X=0
8081 438254;        0206          ,T'C',A(TPSCRN)      ..COPY FROM TAPE TO SCREEN
8084 00;            0207          ,0
8085 ;              0208
8085 ;              0209 ..*************************************************************
8085 ;              0210 ..                    UTILITY ERROR MESSAGE
8085 ;              0211 ..            NOTE: ENTRY HERE RESETS STACK TO TOP
8085 ;              0212
8085 ;              0213 ..REG USED:  CHAR
8085 ;              0214 ..*****************************************************************
8085 F8FFA2;        0215 ERROR:  LDI A.0(TOPSTK);PLO SP
8088 F88CB2;        0216         LDI A.1(TOPSTK);PHI SP
808B D483F0;        0217         SEP CALL;,A(OSTRNG)
808E 0D0A4552524F52;0218         ,A(CRLF),T'ERROR',0
8095 00;            0218
8096 C087F0;        0219 PRMPT1: LBR RENTER
8099 ;              0220
8099 ;              0221 ..*******************************************************
8099 ;              0222 ..                    START OF SUBROUTINES
8099 ;              0223 ..
8099 ;              0224 ..*************************************************************
8099 ;              0225 ..               MONITOR SUBSTITUTE FUNCTION
8099 ;              0226 ..       DISPLAYS THE FIRST BYE FROM THE ADDRESS GIVEN FOLLOWED
8099 ;              0227 ..       BY A HYPHEN. IF A HEX PAIR IS ENTERED FOLLOWED BY A SPACE,
8099 ;              0228 ..       IT IS SUBSTITUTED FOR THE BYTE DISPLAYED, IF A SPACE IS
8099 ;              0229 ..       ENTERED THERE IS NO CHANGE. IN EITHER CASE THE DATA BYTE FROM
8099 ;              0230 ..       THE NEXT ADDRESS WILL THEN BE DISPLAYED. THE ROUTINE IS ENDED
8099 ;              0231 ..       BY ENTERING A RETURN.
8099 ;              0232 ..REG USED:  ASL, SRC, CHAR
8099 ;              0233 ..*********************************************
8099 ;              0234
8099 D487EA;        0235 SUBST:  SEP CALL;,A(READHX)..INPUT ADDRESS
809C 9DBB;          0236         GHI ASL;PHI SRC     ..SAVE STRT ADDRESS
809E 8DAB;          0237         GLO ASL;PLO SRC
80A0 ;              0238
```

```
80A0 9FFB0A;          0239 DECODE: GHI CHAR;XRI LF       ..FIRST NON-HEX MUST BE
80A3 32AF;            0240         BZ ADDOUT ..A LINEFEED OR
80A5 FB07;            0241         XRI #07   ..TERMINATION OR
80A7 3296;            0242         BZ PRMPT1
80A9 FB2D;            0243         XRI #2D   ..A SPACE
80AB 32B7;            0244         BZ OLDDTA
80AD 3085;            0245         BR ERROR          ..ELSE ERROR
80AF ;                0246
80AF D483F00D00;      0247 ADDOUT: SEP CALL;,A(OSTRNG);,CR,0
80B4 D480CD;          0248         SEP CALL;,A(OUT1)
80B7 ;                0249
80B7 DC17;            0250 OLDDTA: SEP DELAY;,#17      ..WAIT TO FINISH READ
80B9 0BBF;            0251         LDN SRC ;PHI CHAR        ..STAY ON SAME LINE
80BB D481AE;          0252         SEP CALL;,A(TYPE2)          ..HEX OUTPUT
80BE D483F0;          0253         SEP CALL;,A(OSTRNG)         ..OUTPUT A HYPHEN
80C1 2D00;            0254         ,T'-',0
80C3 0BAD;            0255         LDN SRC ;PLO ASL..COPY DATA FROM CELL INTO ASL
80C5 ;                0256
80C5 D487EA;          0257 GETDTA: SEP CALL;,A(READHX)        ..GET ANY CHANGE
80C8 8D5B;            0258         GLO ASL;STR SRC    ..RESTORE THE DATA INTO THE CELL
80CA 1B;              0259         INC SRC            ..OPEN THE NEXT CELL
80CB 30A0;            0260         BR DECODE        ..EXAMINE INPUT
80CD ;                0261
80CD DC17;            0262 OUT1:   SEP DELAY;,#17
80CF 9BBF;            0263         GHI SRC;PHI CHAR     ..ROUTINE TO OUTPUT A HEX PAIR
80D1 D481AE;          0264         SEP CALL;,A(TYPE2)          ..AND A SPACE
80D4 8BBF;            0265         GLO SRC;PHI CHAR
80D6 D481AE;          0266         SEP CALL;,A(TYPE2)
80D9 D483F02000;      0267         SEP CALL;,A(OSTRNG);,SPACE,0
80DE D5;              0268         SEP R5
80DF ;                0269
80DF ;                0270 .. ***********************************************************
80DF ;                0271 ..       DELAY IS 2(1 + AUX.1(3 + @LINK)) & IS
80DF ;                0272 ..       USED BY TYPE, READ, AND TIMALC.  AUX.1
80DF ;                0273 ..       IS ASSUMED TO HOLD A DELAY CONSTANT =
80DF ;                0274 ..       ((BIT TIME OF TERMINAL)/
80DF ;                0275 ..       (20 * INSTR TIME OF COSMAC)) - 1.  THIS
80DF ;                0276 ..       CONSTANT CAN BE GENERATED AUTOMATICALLY
80DF ;                0277 ..       BY THE TIMALC ROUTINE.
80DF ;                0278 ..REG USED:  AUX, PC
80DF ;                0279 ..***********************************************************
80DF ;                0280
80DF ;                0281         ORG UT62+#EA
80EA ;                0282
80EA DCDC;            0283 DEXIT:  SEP DELAY; SEP DELAY    ..4 NOP'S
80EC DCDC;            0284         SEP DELAY; SEP DELAY
80EE D3;              0285         SEP PC                  ..RETURN
80EF ;                0286
80EF 9EF6AE;          0287 DELAY1: GHI AUX;SHR;PLO AUX     ..SHIFT OUT ECHO BIT
80F2 ;                0288
80F2 2E;              0289 DELAY2: DEC AUX                 ..AUX.0 = BIT DELAY
80F3 43;              0290         LDA PC                  ..LOAD CONSTANT
80F4 ;                0291
80F4 FF01;            0292 DELAY3: SMI 1                   ..LOOP UNTIL ZERO
80F6 3AF4;            0293         BNZ DELAY3
80F8 8E;              0294         GLO AUX
80F9 32EA;            0295         BZ DEXIT                ..DONE ?
80FB 23;              0296         DEC PC                  ..POINT BACK TO THE CONSTANT
80FC 30F2;            0297         BR DELAY2             ..AND LOOP
80FE ;                0298
```

```
80FE ;              0299
80FE ;              0300 ..***********************************************
80FE ;              0301 ..       CALCULATES BIT TIME AND ECHO FLAG.  WAITS
80FE ;              0302 ..       FOR LF(NO ECHO) OF CR (ECHO) TO BE TYPED
80FE ;              0303 ..       IN.  ALSO SETS UP POINTER TO THE DELAY
80FE ;              0304 ..       ROUTINE.  AUX.1 ENDS UP HOLDING, IN THE
80FE ;              0305 ..       MOST SIGNIFICANT 7 BITS, THE DELAY CONSTANT
80FE ;              0306 ..       THE LEAST SIGNIFICANT BIT IS ZERO FOR ECHO
80FE ;              0307 ..       OR ONE FOR NO ECHO.
80FE ;              0308 ..REG USED:  AUX, CHAR
80FE ;              0309 ..***********************************************
80FE ;              0310
80FE 93BC;          0311 TIMALC: GHI PC;PHI DELAY          ..DELAY SUBROUTINE ADDRESS
8100 F8EFAC;        0312         LDI A.0(DELAY1);PLO DELAY
8103 F800AEAF;      0313         LDI 0;PLO AUX;PLO CHAR
8107 3707;          0314         B4 *            ..WAIT FOR THE START BIT
8109 3F09;          0315         BN4 *           ..WAIT FOR FIRST NON-ZERO
810B F803;          0316         LDI 3           ..DATA BIT.  SET UP FOR 10
810D ;              0317                         ..EXECUTIONS SO THAT THE
810D FF01;          0318 TC2:    SMI 1           ..ROUND-OFF IS MINIMAL
810F 3A0D;          0319         BNZ TC2         ..SEE IF THE DATA CHANGED.
8111 8F;            0320         GLO CHAR        ..BR IF IT HAD, ELSE LOOK
8112 3A17;          0321         BNZ ZERO        ..FOR CHANGE TO ZERO
8114 3719;          0322         B4 INCR         ..BRANCH IF NO
8116 1F;            0323         INC CHAR        ..YES, SET SWITCH
8117 ;              0324
8117 371E;          0325 ZERO:   B4 DAUX    ..LOOK FOR A CHANGE TO 1
8119 ;              0326
8119 1E;            0327 INCR:   INC AUX    ..SET UP FOR 20 INST. LOOPS
811A F807;          0328         LDI 7
811C 300D;          0329         BR TC2
811E ;              0330
811E 2E2E;          0331 DAUX:   DEC AUX; DEC AUX
8120 8EF901BE;      0332         GLO AUX;ORI 1;PHI AUX
8124 DC0C;          0333         SEP DELAY ;,#0C       ..WAIT
8126 3F2C;          0334         BN4 WAIT        ..BR IF LF =] NO ECHO
8128 9EFAFEBE;      0335         GHI AUX;ANI #FE;PHI AUX  ..CR =] ECHO
812C ;              0336
812C DC26;          0337 WAIT:   SEP DELAY ;,#26
812E D5;            0338         SEP R5
812F ;              0339
812F ;              0340 ..***********************************************
812F ;              0341 ..       READS ONE BYTE INTO CHAR.1.  WHEN ENTERED
812F ;              0342 ..       VIA READAH, ANY HEX INPUT IS ASSEMBLED
812F ;              0343 ..       INTO ASL AND DF =1, ELSE DF = 0 ON RETURN.
812F ;              0344 ..       NOTE-DON'T FOLLOW READ WITH IMMEDIATE TYPE.
812F ;              0345 ..       INSERT DELAY OR ENTER TYPE VIA TYPED.
812F ;              0346 ..REG USED:  CHAR, AUX & ASL
812F ;              0347 ..***********************************************
812F ;              0348
812F FC07;          0349 CKDEC:  ADI 7              ..CHECK FOR ASCII DECIMAL
8131 3337;          0350         BDF NFND          ..OUT OF RANGE
8133 FC0A;          0351         ADI #0A             ..SUBTRACT NET 30
8135 337A;          0352         BDF FND
8137 ;              0353
8137 FC00;          0354 NFND:   ADI 0              ..SET DF = 0
8139 ;              0355
8139 9F;            0356 REXIT:  GHI CHAR           ..PUT INPUT INTO D
813A D5;            0357         SEP R5                   .. & RETURN
813B ;              0358
```

```
813B F80038;        0359 READAH: LDI 0;SKP                ..SKIP TO READ1 WITH D=0
813E ;              0360
813E 93;            0361 READ:   GHI PC                   ..CONSTANT GREATER THAN 0
813F ;              0362
813F AF;            0363 READ1:  PLO CHAR                 ..SAVE ENTRY POINT
8140 ;              0364
8140 F880BF;        0365 READ2:  LDI #80; PHI CHAR        ..BIT COUNT, ;SHR; UNTIL DF=1
8143 3F43;          0366         BN4 *                ..WAIT FOR END OF DATA BIT
8145 3745;          0367         B4 *                 ..WAIT FOR START BIT
8147 DC02;          0368         SEP DELAY;,#02           ..HALF BIT TIME DELAY
8149 ;              0369
8149 E2E2;          0370 NOBIT:  SEX SP; SEX SP           ..DELAY
814B ;              0371
814B 9EF6;          0372 BIT:    GHI AUX;SHR                  ..ECHO ?
814D 3357;          0373         BDF READ3           ..BR IF NO
814F 3754;          0374         B4 OUTBIT          ..IS IT A 1 ?
8151 7B;            0375         SEQ                      ..NO
8152 3059;          0376         BR NOECHO
8154 ;              0377
8154 7A;            0378 OUTBIT: REQ                      ..YES
8155 3059;          0379         BR NOECHO
8157 ;              0380
8157 C4C4;          0381 READ3:  NOP; NOP                 ..DELAY
8159 ;              0382
8159 DC07;          0383 NOECHO: SEP DELAY;,#07           ..ONE BIT TIME DELAY
815B C4C4;          0384         NOP; NOP                 ..MORE DELAY
815D 9FF6BF;        0385         GHI CHAR;SHR;PHI CHAR    ..NEXT BIT
8160 3369;          0386         BDF NEXT
8162 F980;          0387         ORI #80
8164 3F49;          0388         BN4 NOBIT          ..BR IF INPUT WAS ZERO
8166 BF;            0389         PHI CHAR
8167 304B;          0390         BR BIT                ..CONTINUE
8169 ;              0391
8169 7A;            0392 NEXT:   REQ                      ..OUTPUT STOP BIT
816A 3240;          0393         BZ READ2           ..IF D=0, CHAR.1 IS A NULL
816C 8F;            0394         GLO CHAR              ..CHECK ENTRY
816D 3A39;          0395         BNZ REXIT          ..ENTERED VIA READ
816F ;              0396
816F 9FFF41;        0397 CKHXE:  GHI CHAR;SMI #41      ..CHECK FOR ASCII HEX
8172 3B2F;          0398         BNF CKDEC          ..CHECK FOR ASCII DECIMAL
8174 FF06;          0399         SMI 6                        ..A THRU F
8176 3337;          0400         BDF NFND           ..NO
8178 FC10;          0401         ADI #10                      ..SUBTRACT NET 37
817A ;              0402
817A AE;            0403 FND:    PLO AUX              ..SAVE TEMPORARILY
817B 9D;            0404         GHI ASL               ..SHIFT DATA INTO ASL
817C FEFEFEFE52;    0405         SHL;SHL;SHL;SHL;STR SP       ..SHL 4X
8181 8D;            0406         GLO ASL
8182 F6F6F6F6;      0407         SHR;SHR;SHR;SHR
8186 F1BD;          0408         OR;PHI ASL
8188 8D;            0409         GLO ASL
8189 FEFEFEFE52;    0410         SHL;SHL;SHL;SHL;STR SP
818E 8EFA0F;        0411         GLO AUX;ANI #0F
8191 F1AD;          0412         OR;PLO ASL
8193 FF00;          0413         SMI 0                        ..SET DF = 1
8195 3039;          0414         BR REXIT
8197 ;              0415
8197 ;              0416 ..*********************************************
8197 ;              0417 ..       TYPES ONE BYTE FROM CHAR.1 AS AN ASCII
8197 ;              0418 ..       CHARACTER OR AS TWO HEX DIGITS.  LINE-FEEDS
```

```
8197 ;              0419 ..       ARE FOLLOWED BY SIX NULLS.  USES REGISTERS
8197 ;              0420 ..       AUX AND CHAR AND A STACK LOCATION.  TYPED
8197 ;              0421 ..       & TYPE5D ALLOW AN EARLIER READ TO COMPLETE.
8197 ;              0422 ..
8197 ;              0423 ..       AUX.0 HOLDS OUTPUT CHARACTER (AT FIRST),
8197 ;              0424 ..       THEN THE DELAY CONSTANT BETWEEN BITS.
8197 ;              0425 ..       CHAR.0 HOLDS THE NUMBER OF BITS (11) IN
8197 ;              0426  ..      ITS LOWER DIGIT AND THE FOLLOWING CODE IN
8197 ;              0427 ..       ITS UPPER DIGIT:
8197 ;              0428 ..             0 -  BYTE OUTPUT
8197 ;              0429 ..             1 -  FIRST HEX OUTPUT
8197 ;              0430 ..             2 -  LAST NULL OUTPUT
8197 ;              0431 ..             8 -  LF OUTPUT
8197 ;              0432 ..
8197 ;              0433 ..REG USED:  CHAR, AUX
8197 ;              0434 ..*************************************************************
8197 ;              0435
8197 ;              0436                ORG UT62+#0198
8198 ;              0437
8198 DC17;          0438 TYPED:  SEP DELAY;,#17       ..THREE BIT TIME DELAY
819A 30A4;          0439         BR TYPE
819C ;              0440
819C DC17;          0441 TYPE5D: SEP DELAY;,#17       ..THREE BIT TIME DELAY
819E 38;            0442         SKP
819F ;              0443
819F D5;            0444 TEXIT:  SEP R5
81A0 ;              0445
81A0 45;            0446 TYPE5:  LDA R5                  ..PICK UP DATA
81A1 38;            0447         SKP
81A2 ;              0448
81A2 46;            0449 TYPE6:  LDA R6                  ..PICK UP DATA
81A3 38;            0450         SKP
81A4 ;              0451
81A4 9FAE;          0452 TYPE:   GHI CHAR;PLO AUX          ..KEEP A COPY
81A6 FB0A;          0453         XRI LF                 ..IS IT A LINE-FEED ?
81A8 3ABF;          0454         BNZ TY5
81AA F88B;          0455         LDI #8B                ..# BITS + # NULLS
81AC 30C1;          0456         BR TY6
81AE ;              0457
81AE 9FF6F6F6F6;    0458 TYPE2:  GHI CHAR;SHR;SHR;SHR;SHR
81B3 FCF6;          0459         ADI #F6                ..CONVERT TO HEX
81B5 3BB9;          0460         BNF TY4          ..IF A OR MORE, ADD 37
81B7 FC07;          0461         ADI 7
81B9 ;              0462
81B9 FFC6AE;        0463 TY4:    SMI #C6;PLO AUX          ..ELSE ADD 30
81BC F81B;          0464         LDI #1B                   ..10 ADI  NO. OF BITS
81BE C8;            0465         LSKP
81BF ;              0466
81BF F80B;          0467 TY5:    LDI #0B                ..NO OF BITS
81C1 ;              0468
81C1 AF;            0469 TY6:    PLO CHAR
81C2 8D73;          0470         GLO ASL;STXD           ..SAVE USER'S RD.0
81C4 ;              0471
81C4 7B;            0472 BEGIN:  SEQ                     ..START BIT
81C5 8EAD;          0473         GLO AUX;PLO ASL          ..COPY CHAR TO RD.0
81C7 ;              0474
81C7 DC07;          0475 PREBIT: SEP DELAY;,#07         ..1 BIT TIME DELAY
81C9 2F;            0476         DEC CHAR                ..DEC BIT COUNT
81CA F5;            0477         SD                      ..SET DF = 1
81CB 8D76AD;        0478         GLO ASL;SHRC;PLO ASL
```

```
81CE 33D3;          0479          BDF OUT1A         ..BR IF BIT IS A ONE
81D0 7B;            0480          SEQ                     ..ELSE OUTPUT A ZERO
81D1 30D5;          0481          BR OUT1B
81D3 ;              0482
81D3 7A7A;          0483 OUT1A:   REQ; REQ                ..OUTPUT A ONE AND DELAY
81D5 ;              0484
81D5 8FFA0F;        0485 OUT1B:   GLO CHAR;ANI #0F        ..DONE ?
81D8 C4C4C4C4;      0486          NOP;NOP;NOP;NOP          ..DELAY
81DC 3AC7;          0487          BNZ PREBIT        ..BR IF NOT
81DE ;              0488
81DE 8FFCFBAF;      0489 NXCHAR:  GLO CHAR;ADI #FB;PLO CHAR
81E2 3BFB;          0490          BNF FIXRD         ..SEP R5 IF NO MORE
81E4 FF1B;          0491          SMI #1B           ..TEST FOR ALTERNATIVES
81E6 32FB;          0492          BZ FIXRD          ..TYPED LAST NULL
81E8 3BEE;          0493          BNF HEX1          ..TYPED FIRST HEX
81EA F800;          0494          LDI 0             ..TYPED LF OR NULL
81EC 30F8;          0495          BR HEX3
81EE ;              0496
81EE 9FFA0F;        0497 HEX1:    GHI CHAR;ANI #0F  ..GET 2ND HEX DIGIT
81F1 FCF6;          0498          ADI #F6           ..CONVERT TO HEX
81F3 C7;            0499          LSNF              ..IF A OR MORE,
81F4 FC07;          0500          ADI 7             ..ADD NET 37
81F6 ;              0501
81F6 FFC6;          0502 HEX2:    SMI #C6           ..ELSE ADD NET 30
81F8 ;              0503
81F8 AE;            0504 HEX3:    PLO AUX           ..AND SAVE
81F9 30C4;          0505          BR BEGIN
81FB ;              0506
81FB 12;            0507 FIXRD:   INC SP
81FC 02AD;          0508          LDN R2;PLO ASL
81FE 309F;          0509          BR TEXIT          ..RESTORE USER'S RD.0
8200 ;              0510
8200 ;              0511 ..*****************************************************
8200 ;              0512 ..                INPUT OPTION
8200 ;              0513 ..       ALLOWS ENTRY OF EITHER STARTING AND ENDING
8200 ;              0514 ..       ADDRESSES OR BYTE COUNT.EXITS WITH THE
8200 ;              0515 ..       STARTING ADDRESS IN REG SRC AND THE BYTE
8200 ;              0516 ..       COUNT IN REG CNT.  RETURNS WITH  DF =1
8200 ;              0517 ..       IF SYNTAX ERROR EXISTS
8200 ;              0518 ..REG USED:  ASL, SRC, CHAR, CNT
8200 ;              0519 ..*****************************************************
8200 ;              0520
8200 D487EA;        0521 OPTION: SEP CALL;,A(READHX)     ..GET THE STARTING ADDRESS
8203 9DBB;          0522         GHI ASL;PHI SRC         ..AND SAVE IT
8205 8DAB;          0523         GLO ASL;PLO SRC
8207 F800ADBD;      0524         LDI 0;PLO ASL;PHI ASL   ..CLEAR THE INPUT REG.
820B 9FFB20;        0525         GHI CHAR;XRI SPACE       ..FIRST NONSMI HEX MUST
820E 3231;          0526         BZ CNTIN          ..BE A SPACE OR
8210 FB0D;          0527         XRI #0D           ..A HYPHEN
8212 3A46;          0528         BNZ PRMPT2        ..ELSE SYNTAX ERROR
8214 D487EA;        0529         SEP CALL;,A(READHX) ..EXPECT ENDING ADDRESS
8217 ;              0530
8217 8B52;          0531 BYTCNT: GLO  SRC;STR SP              ..CALCULATE THE BYTE COUNT
8219 8DF7AA;        0532         GLO ASL;SM;PLO CNT
821C 9B52;          0533         GHI SRC;STR SP
821E 9D77BA;        0534         GHI ASL;SMB;PHI CNT
8221 333F;          0535         BDF EXITOK          ..CHECK FOR SRC LESS THAN  ASL
8223 ;              0536
8223 8D52;          0537 INVERT: GLO ASL;STR SP          ..ELSE EXCHANGE THE CONTENTS OF
8225 8BAD;          0538         GLO SRC;PLO ASL         ..SRC AND ASL
```

```
8227 02AB;          0539         LDN SP;PLO SRC
8229 9D52;          0540         GHI ASL;STR SP
822B 9BBD;          0541         GHI SRC;PHI ASL
822D 02BB;          0542         LDN SP;PHI SRC
822F 3017;          0543         BR  BYTCNT           ..RECALCULATE
8231 ;              0544
8231 D487EA;        0545 CNTIN:  SEP CALL;,A(READHX)  ..INPUT THE BYTE COUNT
8234 8DFF01AA;      0546         GLO ASL;SMI 1;PLO CNT
8238 9D7F00BA;      0547         GHI ASL;SMBI 0;PHI CNT
823C 333F;          0548         BDF EXITOK
823E 1A;            0549         INC CNT
823F D5;            0550 EXITOK: SEP R5                  ..RETURN WHEN DONE
8240 ;              0551
8240 ;              0552 ..*****************************************************
8240 ;              0553 ..                    FILL ROUTINE
8240 ;              0554 ..       LOADS MEMORY BEGINNING AT ADDRESS CONTAINED
8240 ;              0555 ..       IN SRC WITH DATA CONTAINED IN ASL.0 FOR
8240 ;              0556 ..       THE NUMBER OF BYTES SPECIFIED BY CNT.
8240 ;              0557 ..       USER CALLABLE @ USRFIL.
8240 ;              0558 ..REG USED:  ASL, SRC, CNT ,CHAR
8240 ;              0559 ..*****************************************************
8240 ;              0560
8240 D48303;        0561 FILL:   SEP CALL;,A(READAD)     ..GET THE ADDRESSES
8243 D4824B;        0562         SEP CALL;,A(USRFIL)     ..CALL THE MOVE
8246 ;              0563
8246 C087F0;        0564 PRMPT2: LBR RENTER              ..GOTO UT62 AND PROMPT
8249 ;              0565
8249 1B;            0566 NXTCEL: INC SRC                 ..POINT TO NEXT CELL
824A 2A;            0567         DEC CNT                 ..REDUCE BYTE COUNT
824B 8D5B;          0568 USRFIL: GLO ASL;STR SRC         ..LOAD THE DATA;USER ENTRY PT.
824D 8A;            0569         GLO CNT                 ..LOOP UNTIL COUNT = 0
824E 3A49;          0570         BNZ NXTCEL
8250 9A;            0571         GHI CNT
8251 3A49;          0572         BNZ NXTCEL
8253 D5;            0573         SEP R5                  ..EXIT THE CALL
8254 ;              0574
8254 ;              0575 ..*****************************************************
8254 ;              0576 ..            ROUTINE TO READ FROM TAPE TO SCREEN
8254 ;              0577 ..       AFTER EACH 20 RETURNS, ROUTINE OUTPUTS "****" AND
8254 ;              0578 ..       WAITS FOR BREAK KEY TO BE HIT TO CONTINUE.
8254 ;              0579 ..       HALTS ON BREAK, EXITS ON Q OR CONTINUE ON OTHER
8254 ;              0580 ..*****************************************************
8254 D48774;        0581 TPSCRN: SEP CALL;,A(FINDTP)     ..GET TAPE #
8257 9C73;          0582         GHI RC;STXD             ..SAVE IOCB
8259 8C73;          0583         GLO RC;STXD
825B D483F00A00;    0584         SEP CALL;,A(OSTRNG),LF,0..OUTPUT A LINEFEED
8260 12;            0585         INC SP                  ..RESTORE IOCB
8261 42AC;          0586         LDA SP;PLO RC
8263 02BC;          0587         LDN SP;PHI RC
8265 F814AA;        0588         LDI LINES;PLO CNT       ..INITIALIZE FOR 20 LINES
8268 ;              0589
8268 D48542;        0590 MORLST: SEP CALL;,A(READT)      ..GET A BYTE
826B ;              0591
826B 9C73;          0592 CHKOUT: GHI RC;STXD
826D 8C73;          0593         GLO RC;STXD
826F 9FFB13;        0594         GHI CHAR;XRI EOF
8272 3246;          0595         BZ PRMPT2               ..WE'RE DONE
8274 FB1E;          0596         XRI #1E                 ..ASSEMBLY LISTINGS DON'T HAVE
8276 3AA4;          0597         BNZ SHOWIT
8278 D483F0;        0598         SEP CALL;,A(OSTRNG)        ..LINE FEEDS
```

```
827B 0D0A00;          0599          ,A(CRLF),0
827E 2A8A;            0600          DEC CNT;GLO CNT             ..DID 20 RETURNS GO BY ?
8280 3A96;            0601          BNZ NXTLNE
8282 D483F0;          0602          SEP CALL;,A(OSTRNG)
8285 2A2A2A2A00;      0603          ,T'****',0              ..PROMPT FOR A KEY WHEN READY
828A D4813E;          0604          SEP CALL;,A(READ)
828D D483F0;          0605          SEP CALL;,A(OSTRNG)
8290 0D0A00;          0606          ,A(CRLF),0
8293 F814AA;          0607          LDI LINES;PLO CNT
8296 ;                0608
8296 12;              0609 NXTLNE: INC SP
8297 72AC;            0610          LDXA;PLO RC
8299 02BC;            0611          LDN SP;PHI RC
829B D48542;          0612          SEP CALL;,A(READT)
829E FB0A;            0613          XRI LF
82A0 3268;            0614          BZ MORLST
82A2 ;                0615                                  ..IT ISN'T A LINE FEED SO
82A2 306B;            0616          BR CHKOUT               ..SEE WHAT IT IS
82A4 ;                0617
82A4 F8EFAC;          0618 SHOWIT: LDI A.0(DELAY1);PLO DELAY
82A7 F880BC;          0619          LDI A.1(DELAY1);PHI DELAY
82AA D481A4;          0620          SEP CALL;,A(TYPE)       ..PUT IT TO THE TERMINAL
82AD 37B6;            0621          B4 MRLST1       ..HALT ON BREAK ASSERTED
82AF D4813E;          0622          SEP CALL;,A(READ)
82B2 FB51;            0623          XRI T'Q'        ..EXIT ON Q OR CONTINUE ON
82B4 3246;            0624          BZ PRMPT2       ..ANYTHING ELSE
82B6 ;                0625
82B6 12;              0626 MRLST1: INC SP                   ..RESTORE IOCB POINTER
82B7 72AC;            0627          LDXA;PLO RC
82B9 02BC;            0628          LDN SP;PHI RC
82BB 3068;            0629          BR MORLST
82BD ;                0630
82BD ;                0631 ..*****************************************************************
82BD ;                0632 ..                       OUTPUT
82BD ;                0633 ..       FORMATS AND OUTPUTS MEMORY DATA BEGINNING
82BD ;                0634 ..       AT THE  ADDRESS IN REG SRC FOR THE NUMBER
82BD ;                0635 ..       OF BYTES SPECIFIED IN REG CNT
82BD ;                0636 ..REG USED:  SRC, CNT, CHAR
82BD ;                0637 ..*****************************************************************
82BD ;                0638
82BD D48200;          0639 DISPLY: SEP CALL;,A(OPTION)      ..GET STARTING ADDRESS
82C0 FB0D;            0640          XRI CR                  ..TERMINATE WITH CR
82C2 CA8085;          0641          LBNZ ERROR
82C5 ;                0642
82C5 D483F00A00;      0643 OUTPUT: SEP CALL;,A(OSTRNG);,LF,0..START ON A NEW LINE
82CA D480CD;          0644          SEP CALL;,A(OUT1)       ..OUTPUT THE ADDRESS
82CD ;                0645                                  ..OF THE CURRENTLY OPENED
82CD ;                0646                                  ..CELL
82CD ;                0647
82CD D483F0;          0648 SPCOUT: SEP CALL;,A(OSTRNG)
82D0 2000;            0649          ,SPACE,0
82D2 ;                0650
82D2 4BBF;            0651 DATOUT: LDA SRC;PHI CHAR         ..RETRIEVE THE CELL DATA
82D4 D481AE;          0652          SEP CALL;,A(TYPE2)      ..AND OUTPUT IT
82D7 8A;              0653          GLO CNT                 ..DETERMINE IF THE
82D8 3ADD;            0654          BNZ NOTDON              ..REQUESTED NO. OF BYTES
82DA 9A;              0655          GHI CNT                 ..HAVE BEEN SENT
82DB 3246;            0656          BZ PRMPT2               ..GET A NEW COMMAND
82DD ;                0657
82DD 2A;              0658 NOTDON: DEC CNT                  ..DEC THE BYTE COUNT
```

```
82DE 8BFA0F;        0659         GLO SRC;ANI LNECNT
82E1 3AEB;          0660         BNZ SAMELN              ..END OF CURRENT LINE?
82E3 D483F0;        0661         SEP CALL;,A(OSTRNG)
82E6 3B0D00;        0662         ,T';',CR,0
82E9 30C5;          0663         BR OUTPUT
82EB ;              0664
82EB F6;            0665 SAMELN: SHR
82EC 33D2;          0666         BDF DATOUT             ..WITHIN PAIR
82EE 30CD;          0667         BR  SPCOUT             ..ELSE BETWEEN PAIRS
82F0 ;              0668
82F0 ;              0669 ..*****************************************************************
82F0 ;              0670 ..      READ 1 CHARACTER FROM TAPE AND DF=1 IF HEX
82F0 ;              0671 ..*****************************************************************
82F0 ;              0672
82F0 D48542;        0673 RDHEX:  SEP CALL;,A(READT)
82F3 D4816F;        0674         SEP CALL;,A(CKHXE)
82F6 D5;            0675         SEP R5
82F7 ;              0676
82F7 ;              0677 ..*****************************************************************
82F7 ;              0678 ..              UT62 MOVE COMMAND
82F7 ;              0679 ..              CALLS USRMOV AND REQUESTS SRC&DEST ADDR'S
82F7 ;              0680 ..*****************************************************************
82F7 D48303;        0681 MOVE:   SEP CALL;,A(READAD)     ..GET SRC&DEST ADDR'S
82FA D48429;        0682         SEP CALL;,A(USRMOV)     ..DO THE MOVE
82FD C38085;        0683         LBDF ERROR              ..ERROR IF OVER FFFF ON MOVE
8300 C08246;        0684         LBR PRMPT2               ..IF OK,GOTO UT62 PROMPT
8303 ;              0685 ..*******************************************************
8303 ;              0686 ..    SUBROUTINE TO GET THE ADDRESSES FOR OTHER ROUTINES
8303 ;              0687 ..*******************************************************
8303 D48200;        0688 READAD: SEP CALL;,A(OPTION)     ..DETERMINE THE MODE
8306 FB20;          0689         XRI SPACE               ..MUST BE A SPACE
8308 3A60;          0690         BNZ ERR1                ..ELSE ERROR
830A ADBD;          0691         PLO ASL;PHI ASL         ..CLEAR INPUT REGISTER
830C D487EA;        0692         SEP CALL;,A(READHX)     ..INPUT THE CONSTANT
830F FB0D;          0693         XRI CR                  .. 'CR' TERMINATES
8311 3A60;          0694         BNZ ERR1                ..ELSE ERROR
8313 D5;            0695         SEP R5
8314 ;              0696 ..*****************************************************
8314 ;              0697 ..      FILLS ASL UNTIL A CARRIAGE RETURN IS ENTERED
8314 ;              0698 ..*****************************************************
8314 D4813B;        0699 READCR: SEP CALL;,A(READAH)
8317 FB0D;          0700         XRI CR
8319 3A14;          0701         BNZ READCR
831B D5;            0702         SEP R5
831C ;              0703 ..*****************************************************************
831C ;              0704 ..            READ A BIT OF DATA FROM TAPE
831C ;              0705 ..            ENTERED WITH SET P TO BITRDR
831C ;              0706 ..*****************************************************************
831C ;              0707
831C D3;            0708 RBYE:   SEP PC
831D ;              0709
831D F816;          0710 RDBIT:  LDI #16          ..0/1 BIT DISTINGUISH TIMER
831F ;              0711
831F ;              0712 ..*****************************************************************
831F ;              0713 ..                              FOR 2 MHZ CRYSTAL
831F ;              0714 ..                      THE TIMING LOOP USED GIVES 22X24US DELAY(528)
831F ;              0715 ..                      TO TEST FOR A 1/0 TRANSITION DATA FROM TAPE
831F ;              0716 ..                      TYPICALLY HAS A 350/750US DIFFERENCE. (0/1)
831F ;              0717 ..*****************************************************************
831F ;              0718
```

```
831F 3D1F;          0719          BN2 *           ..WAIT FOR TRANSITION
8321 D8;            0720          SEP BITRDR      ..DELAY AND THEN RETEST
8322 3D1D;          0721          BN2 RDBIT       ..TO BE SURE
8324 3D34;          0722 HERE:    BN2 SLIDE1      ..TIME OUT LOOP
8326 FF013A24;      0723 HERE1:   SMI 1; BNZ HERE
832A ;              0724
832A 17;            0725 HERE2:   INC PARITY      ..SET BIT READ AS A ONE
832B 93;            0726          GHI PC
832C ;              0727
832C FE;            0728 SLIDE2:  SHL             ..PUT BIT IN DF
832D 352D;          0729 SLIDE3:  B2 *            ..WAIT FOR BIT TO END
832F D8;            0730          SEP BITRDR      ..DELAY AND RETEST TO
8330 352D;          0731          B2 SLIDE3       ..BE SURE
8332 301C;          0732          BR RBYE
8334 ;              0733
8334 FF01;          0734 SLIDE1:  SMI 1           ..DECREMENT LOOP COUNT AND RETEST
8336 3D2C;          0735          BN2 SLIDE2      ..BRANCH IF STILL FALSE TO SET BIT A ZERO
8338 3A26;          0736          BNZ HERE1       ..IF TIME OUT GO TO SET BIT A ONE
833A 302A;          0737          BR HERE2        ..ELSE CONTINUE TIMEOUT
833C ;              0738
833C ;              0739 ..*************************************************************
833C ;              0740                ORG UT62+#035D
835D ;              0741
835D C087F0;        0742 PRMPT5:  LBR RENTER
8360 C08085;        0743 ERR1:    LBR ERROR             ..GENERAL FOR THIS PAGE
8363 ;              0744
8363 ;              0745 ..*************************************************************
8363 ;              0746 ..DESC:  STANDARD CALL AND RETURN
8363 ;              0747 ..REG USED:  SP,PC,CALL,RETURN,LINK & STACK
8363 ;              0748 ..*************************************************************
8363 ;              0749                  ORG UT62+#0363
8363 ;              0750                .. STANDARD CALL
8363 ;              0751
8363 D3;            0752 EXITC:   SEP PC                ..GO TO IT
8364 ;              0753
8364 E2;            0754 CALLR:   SEX SP                ..SET R(X)
8365 9673;          0755          GHI LINK;STXD         ..SAVE THE CURRENT LINK ON
8367 8673;          0756          GLO LINK;STXD         ..THE STACK
8369 93B6;          0757          GHI PC;PHI LINK
836B 83A6;          0758          GLO PC;PLO LINK
836D 46B3;          0759          LDA LINK;PHI PC       ..PICK UP THE SUBROUTINE
836F 46A3;          0760          LDA LINK;PLO PC       ..ADDRESS
8371 3063;          0761          BR EXITC
8373 ;              0762                  ..STANDARD RETURN
8373 ;              0763
8373 D3;            0764 EXITR:   SEP PC                ..RETURN TO MAIN PGM.
8374 ;              0765
8374 96B3;          0766 RETR:    GHI LINK;PHI PC
8376 86A3;          0767          GLO LINK;PLO PC
8378 E212;          0768          SEX SP;INC SP         ..SET THE STACK POINTER
837A 72A6;          0769          LDXA;PLO LINK         ..RESTORE THE CONTENTS OF
837C F0B6;          0770          LDX;PHI LINK        ..LINK
837E 9F;            0771          GHI CHAR              ..PUT THE CONTENTS OF CHAR.1 INTO D
837F ;              0772                                ..BEFORE RETURNING
837F 3073;          0773          BR EXITR
8381 ;              0774
8381 ;              0775 ..*************************************************************************
8381 ;              0776 ..              REGISTER INITIALIZATION ROUTINE
8381 ;              0777 ..      INITIALIZES REGISTER C TO THE DELAY ROUTINE, REG 2 AS A STACK
8381 ;              0778 ..      POINTER TO LOCATION 8CFF HEX, REG 4 TO CALL, REG 5 TO RETURN
```

```
8381 ;              0779 ..      AND REG 3 AS PROGRAM COUNTER. FOR ENTER1 REG 3 IS 0005, FOR
8381 ;              0780 ..      ENTER2 REG 3 MUST BE PRESET.
8381 ;              0781 ..      REG USED:  PC, DELAY, CALL, RETURN, SP
8381 ;              0782 ..*********************************************************************
8381 ;              0783
8381 F82CA3;        0784 INIT:   LDI A.0(START);PLO PC
8384 F880B3;        0785         LDI A.1(START);PHI PC
8387 308F;          0786         BR ENTER2
8389 ;              0787
8389 F805A3;        0788 ENTER1: LDI A.0(PGMSRT);PLO PC
838C F800B3;        0789         LDI A.1(PGMSRT);PHI PC
838F ;              0790
838F F8EFAC;        0791 ENTER2: LDI A.0(DELAY1);PLO DELAY   ..DELAY ROUTINE
8392 F880BC;        0792         LDI A.1(DELAY1);PHI DELAY
8395 F883B4B5;      0793         LDI A.1(CALLR);PHI CALL;PHI RETN
8399 F864A4;        0794         LDI A.0(CALLR);PLO CALL
839C F874A5;        0795         LDI A.0(RETR);PLO RETN
839F F8FFA2;        0796         LDI A.0(TOPSTK);PLO SP
83A2 F88CB2;        0797         LDI A.1(TOPSTK);PHI SP
83A5 E2D3;          0798         SEX SP;SEP PC
83A7 ;              0799
83A7 ;              0800 ..*********************************************************
83A7 ;              0801 ..              HEX BYTE INSERT ROUTINE
83A7 ;              0802 ..      INSERTS HEX PAIRS INTO MEMORY STARTING AT A SPECIFIED
83A7 ;              0803 ..      ADDRESS. AFTER A ";" ALL IS IGNORED UNTIL A RETURN
83A7 ;              0804 ..      THEN A NEW ADDRESS IS EXPECTED. ANY NON-HEX DATA IS
83A7 ;              0805 ..      IGNORED BETWEEN HEX PAIRS BUT NOTHING IS PERMITTED
83A7 ;              0806 ..      BETWEEN MEMBERS OF THE PAIR. ROUTINE IS TERMINATED
83A7 ;              0807 ..      WITH A RETURN, EXCEPT AFTER A ";".
83A7 ;              0808 ..      REG USED:  ASL, SRC, CHAR
83A7 ;              0809 ..*********************************************************
83A7 ;              0810
83A7 F800BDAD;      0811 INSERT: LDI 0;PHI ASL;PLO ASL   ..CLEAR INPUT REGISTER
83AB ;              0812
83AB D487EA;        0813 FSTHEX: SEP CALL;,A(READHX)     ..WAIT FOR FIRST NON-HEX INPUT
83AE FB20;          0814         XRI SPACE               ..FIRST NON-HEX MUST BE
83B0 3A60;          0815         BNZ ERR1                ..A SPACE
83B2 9DBB;          0816         GHI ASL;PHI SRC         ..STORE STARTING ADDRESS
83B4 8DAB;          0817         GLO ASL;PLO SRC
83B6 ;              0818
83B6 D4813B;        0819 NXTCHR: SEP CALL;,A(READAH)     ..GET FIRST CHARACTER
83B9 3BC5;          0820         BNF NTDATA
83BB D4813B;        0821         SEP CALL;,A(READAH)     ..OTHERWISE SECOND ONE MUST BE HEX
83BE 3B60;          0822         BNF ERR1
83C0 8D5B;          0823         GLO ASL;STR SRC         ..STORE THE DATA
83C2 1B;            0824         INC SRC                 ..POINT TO THE NEXT CELL
83C3 30B6;          0825         BR NXTCHR            .. & CONTINUE
83C5 ;              0826
83C5 FB0D;          0827 NTDATA: XRI CR                  ..DONE IF CR
83C7 325D;          0828         BZ PRMPT5
83C9 9F;            0829         GHI CHAR
83CA FB3B;          0830         XRI SEMCOL              ..IF ";" WAIT FOR CRLF
83CC 3AD3;          0831         BNZ COMCHK              ..ELSE CHECK IF ","
83CE D483DD;        0832         SEP CALL;,A(CKCRLF)
83D1 30A7;          0833         BR INSERT               ..GO GET NEW ADDRESS
83D3 ;              0834
83D3 9F;            0835 COMCHK: GHI CHAR                ..CHECK FOR ","
83D4 FB2C;          0836         XRI COMMA
83D6 3AB6;          0837         BNZ NXTCHR              ..IGNORE IF NOT , ; CR
83D8 D483DD;        0838         SEP CALL;,A(CKCRLF)     ..WAIT FOR CRLF
```

```
83DB 30B6;          0839         BR NXTCHR
83DD ;              0840
83DD D4813E;        0841 CKCRLF: SEP CALL;,A(READ)       ..GET A CHAR
83E0 FB0D;          0842         XRI CR                  ..IS IT CR?
83E2 3A60;          0843         BNZ ERR1                ..BETTER BE!
83E4 D4813E;        0844         SEP CALL;,A(READ)       ..GET NEXT CHAR
83E7 FB0A;          0845         XRI LF                  ..IS IT LF?
83E9 3A60;          0846         BNZ ERR1                ..ERROR IF NO!
83EB D5;            0847         SEP R5                  ..EXIT IF ALL OK
83EC ;              0848
83EC ;              0849 ..*************************************************
83EC ;              0850 ..              UTILITY ENTRY TABLE
83EC ;              0851 ..*************************************************
83EC ;              0852            ORG UT62+#03F0
83F0 ;              0853
83F0 C08419;        0854 OSTRNG: LBR MSGE
83F3 C08389;        0855 INIT1:  LBR ENTER1
83F6 C0838F;        0856 INIT2:  LBR ENTER2
83F9 C087F0;        0857 GOUT62: LBR RENTER
83FC C0816F;        0858 CKHEX:  LBR CKHXE
83FF ;              0859
83FF ;              0860 ..*****************************************************
83FF ;              0861                 ..TAPES OFF ROUTINE
83FF E3;            0862 TPOFF:  SEX PC
8400 6102;          0863         OUT BDSEL;,TAPEIO       ..SELECT TAPE BD
8402 6400;          0864         OUT TPESEL;,TPEOFF      ..OUTPUT OFF BITS
8404 6101;          0865         OUT BDSEL;,TRMINL       ..SELECT RCA GRP
8406 D5;            0866         SEP R5                  ..RETURN TO CALLER
8407 ;              0867
8407 ;              0868 ..*************************************************************
8407 ;              0869 ..              ..REGISTER SAVE ROUTINES
8407 ;              0870 ..            SAVES THE CONTENTS OF REGISTERS TMPRG1 (7),
8407 ;              0871 ..            TMPRG2 (8), TMPRG3 (9) AND CNT (A) ON THE STACK.
8407 ;              0872 ..            USES 8 BYTES OF STACK, AND LEAVES STACK POINTER
8407 ;              0873 ..            AT A FREE BYTE
8407 ;              0874 ..*************************************************************
8407 ;              0875
8407 8A73;          0876 REGSAV: GLO CNT;STXD
8409 9A73;          0877         GHI CNT;STXD
840B 8973;          0878         GLO TMPRG3;STXD
840D 9973;          0879         GHI TMPRG3;STXD
840F 8873;          0880         GLO TMPRG2;STXD
8411 9873;          0881         GHI TMPRG2;STXD
8413 8773;          0882         GLO TMPRG1;STXD
8415 9773;          0883         GHI TMPRG1;STXD
8417 9F;            0884         GHI CHAR        ..TO COMPENSATE FOR BASIC CALL
8418 D5;            0885         SEP R5
8419 ;              0886
8419 ;              0887 ..*************************************************************
8419 ;              0888 ..              OSTRNG
8419 ;              0889 ..*************************************************************
8419 ;              0890
8419 F8EFAC;        0891 MSGE:   LDI #EF;PLO DELAY
841C F880BC;        0892         LDI #80;PHI DELAY
841F ;              0893
841F 46BF;          0894 MSGE1:  LDA LINK;PHI CHAR
8421 3228;          0895         BZ EXITM
8423 D48198;        0896         SEP CALL;,A(TYPED)
8426 301F;          0897         BR MSGE1
8428 ;              0898
```

```
8428 D5;           0899 EXITM:  SEP R5
8429 ;             0900
8429 ;             0901 ..*************************************************************
8429 ;             0902 ..              MOVE ROUTINE
8429 ;             0903 ..       COPIES A BLOCK OF MEMORY FORM ONE CONTINUOUS AREA
8429 ;             0904 ..       TO ANOTHER CONTINUOUS AREA IN MEMORY. THERE IS NO
8429 ;             0905 ..       RESTRICTION AS TO THE DIRECTION OF THE MOVE AND THE
8429 ;             0906 ..       AREAS MAY OVERLAP.
8429 ;             0907 ..       REG USED:  SRC, DEST, CHAR, & CNT
8429 ;             0908 ..*************************************************************
8429 ;             0909
8429 E2;           0910 USRMOV: SEX SP
842A 8B52;         0911         GLO SRC;STR SP          ..TEST THE RELATIVE POSITION
842C 8DF7;         0912         GLO DEST;SM             ..OF SOURCE & DESTINATION
842E 3A36;         0913         BNZ DIRECT              ..NOT EQUAL!
8430 9B52;         0914         GHI SRC;STR SP          ..RETURN IF THEY ARE EQUAL
8432 9D77;         0915         GHI DEST;SMB
8434 3272;         0916         BZ USRBYE               ..EXIT TO CALLER
8436 ;             0917
8436 8B52;         0918 DIRECT: GLO SRC;STR SP          ..ELSE TEST FOR UP OR DOWN
8438 8DF7;         0919         GLO DEST;SM             ..DIRECTION OF THE MOVE
843A 9B52;         0920         GHI SRC;STR SP
843C 9D77;         0921         GHI DEST;SMB
843E 334D;         0922         BDF MOVUP
8440 ;             0923
8440 0B5D;         0924 MOVDN:  LDN SRC;STR DEST        ..DO THE MOVE DOWN AND
8442 8A;           0925         GLO CNT                 ..AND CHECK IF DONE
8443 3A48;         0926         BNZ MOVDN1
8445 9A;           0927         GHI CNT
8446 3272;         0928         BZ USRBYE               ..EXIT TO CALLER
8448 ;             0929
8448 1B1D;         0930 MOVDN1: INC SRC;INC DEST        ..ADJUST THE POINTERS
844A 2A;           0931         DEC CNT                 ..REDUCE THE BYTE COUNT
844B 3040;         0932         BR MOVDN                ..FINISHED
844D ;             0933
844D 8A52;         0934 MOVUP:  GLO  CNT;STR SP         ..SET THE POINTERS TO THE
844F 8BF4AB;       0935         GLO SRC;ADD;PLO SRC     ..TOP OF MOVE AREAS
8452 9A52;         0936         GHI CNT;STR SP
8454 9B74BB;       0937         GHI SRC;ADC;PHI SRC
8457 8A52;         0938         GLO CNT;STR SP
8459 8DF4AD;       0939         GLO DEST;ADD;PLO DEST
845C 9A52;         0940         GHI CNT;STR SP
845E 9D74BD;       0941         GHI DEST;ADC;PHI DEST
8461 3B65;         0942         BNF UP
8463 ;             0943
8463 3073;         0944 ERRGO:  BR USRBYE+1             ..EXIT DF=1 IF OVERFLOW
8465 ;             0945
8465 0B5D;         0946 UP:     LDN SRC;STR DEST        ..DO THE MOVE UP
8467 8A;           0947         GLO CNT                 ..AND CHECK IF DONE
8468 3A6D;         0948         BNZ UP1
846A 9A;           0949         GHI CNT
846B 3272;         0950         BZ USRBYE               ..EXIT TO CALLER
846D ;             0951
846D 2B2D2A;       0952 UP1:    DEC SRC;DEC DEST;DEC CNT..ADJUST THE POINTERS
8470 3065;         0953         BR UP
8472 ;             0954
8472 F6;           0955 USRBYE: SHR                     ..SET DF=0 IF A[FFFF
8473 D5;           0956         SEP R5                  ..EXIT TO CALLER
8474 ;             0957
8474 ;             0958 ..*************************************************************
```

```
8474 ;              0959 ..              TAPE LOADER ROUTINE
8474 ;              0960 ..LOADS USER MEMORY FROM TAPE, ASSUMES TAPE HAS "M" OR "I" TO
8474 ;              0961 ..INDICATE START OF DATA TO BE STORED. "P" FOLLOWED BY AN ADDRESS
8474 ;              0962 ..STARTS PROGRAM EXECUTION WITH P=0 AND X=0.
8474 ;              0963 ..************************************************************
8474 ;              0964
8474 D48774;        0965 TPLOAD: SEP CALL;,A(FINDTP)       ..SELECT TAPE
8477 9C73;          0966         GHI RC;STXD
8479 8C73;          0967         GLO RC;STXD
847B D483F0;        0968         SEP CALL;,A(OSTRNG)
847E 0A4C4F4144494E;0969         ,LF,T'LOADING',0
8485 4700;          0969
8487 12;            0970         INC SP
8488 72AC;          0971         LDXA;PLO RC
848A 02BC;          0972         LDN SP;PHI RC
848C ;              0973
848C D48542;        0974 READZ:  SEP CALL;,A(READT)      ..READ 1 ASCII DIGIT
848F FB50;          0975         XRI T'P'                ..CHECK FOR PROGRAM EXECUTE
8491 32DF;          0976         BZ ADLP
8493 FB1D;          0977         XRI #1D                 ..SET MEMORY?
8495 CE;            0978         LSZ
8496 FB04;          0979         XRI #04                 ..INSERT MEMORY?
8498 32A0;          0980         BZ READX1
849A FB5A;          0981         XRI #5A                 ..CHECK FOR EOF(DC3)
849C 32F9;          0982         BZ PRMPT4
849E 308C;          0983         BR READZ
84A0 ;              0984
84A0 ;              0985                 ..READ THE HEX ADDRESS
84A0 ;              0986
84A0 D482F0;        0987 READX1: SEP CALL;,A(RDHEX) ..IGNORE ALL TILL HEX DIGIT
84A3 33B2;          0988         BDF READX2          ..THEN GET SECOND ONE
84A5 FB2E;          0989         XRI #2E             ..UNLESS "." WHICH MEANS COMMENT
84A7 3AA0;          0990         BNZ READX1
84A9 D48542;        0991 READXA: SEP CALL;,A(READT)  ..THEN IGNORE ALL TILL END OF LINE
84AC FB0D;          0992         XRI CR
84AE 3AA9;          0993         BNZ READXA
84B0 30A0;          0994         BR READX1
84B2 ;              0995
84B2 D482F0;        0996 READX2: SEP CALL;,A(RDHEX) ..READ NEXT ASCII CHARACTER
84B5 33B2;          0997         BDF READX2        ..IF HEX KEEP READING
84B7 FB20;          0998         XRI SPACE         ..OTHERWISE MUST BE A SPACE
84B9 3AE7;          0999         BNZ ERRORX
84BB 9DB8;          1000         GHI ASL;PHI ADRPTR
84BD 8DA8;          1001         GLO ASL;PLO ADRPTR
84BF ;              1002
84BF D482F0;        1003 READX3: SEP CALL;,A(RDHEX) ..READ NEXT ASCII CHARACTER
84C2 33CE;          1004         BDF READXB        ..IF HEX THEN IT'S DATA & NEED ANOTHER
84C4 FB0D;          1005         XRI CR            ..OR COULD BE END OF LINE
84C6 328C;          1006         BZ READZ          ..THEN START AGAIN
84C8 FB36;          1007         XRI #36           ..OR IF ";" THEN IT IS END OF DATA
84CA 32A9;          1008         BZ READXA         ..FOR THIS LINE
84CC 30BF;          1009         BR READX3              ..CAN ONLY BE MORE DATA
84CE ;              1010
84CE D482F0;        1011 READXB: SEP CALL;,A(RDHEX) ..READ 2ND ASCII CHARACTER
84D1 3BE7;          1012         BNF ERRORX          ..BETTER BE A HEX DIGIT
84D3 8D58;          1013         GLO ASL;STR ADRPTR  ..STORE AT THE SPECIFIED ADDRESS
84D5 E8F3;          1014         SEX ADRPTR;XOR      ..MAKE SURE THERE WAS RAM THERE
84D7 32DC;          1015         BZ WRTOK
84D9 D485C5;        1016         SEP CALL;,A(NOTRAM)
84DC ;              1017
```

```
84DC 18;            1018 WRTOK:  INC ADRPTR
84DD 30BF;          1019         BR READX3
84DF ;              1020
84DF ;              1021                 ..ROUTINE TO START A PROGRAM RUNNING
84DF ;              1022
84DF D482F0;        1023 ADLP:   SEP CALL;,A(RDHEX)  ..GET THE START ADDRESS
84E2 33DF;          1024         BDF ADLP
84E4 C0853C;        1025         LBR RUN1
84E7 ;              1026
84E7 ;              1027                 ..ERROR MESSAGE
84E7 ;              1028
84E7 D483F0;        1029 ERRORX: SEP CALL;,A(OSTRNG)
84EA 0D0A464F524D41;1030         ,A(CRLF),T'FORMAT ERROR',0
84F1 54204552524F52;1030
84F8 00;            1030
84F9 C087F0;        1031 PRMPT4: LBR RENTER
84FC ;              1032
84FC ;              1033 ..*************************************************************
84FC ;              1034 ..      USER ENTRY POINTS FOR TAPE ROUTINES
84FC ;              1035 ..*************************************************************
84FC ;              1036                   ORG #8500
8500 ;              1037
8500 3090;          1038         BR TWRITE             ..ENTRY TO TAPE WRITE
8502 ;              1039
8502 3042;          1040         BR READT              ..ENTRY TO TAPE READ
8504 ;              1041
8504 ;              1042 ..*************************************************************
8504 ;              1043 ..              TAPE REWIND ROUTINE
8504 ;              1044 ..*************************************************************
8504 ;              1045
8504 D483F0;        1046 REWIND: SEP CALL;,A(OSTRNG)
8507 0D0A524557494E;1047         ,A(CRLF),T'REWIND,THEN HIT ANY KEY',0
850E 442C5448454E20;1047
8515 48495420414E59;1047
851C 204B455900;    1047
8521 E3;            1048         SEX PC
8522 6102;          1049         OUT BDSEL;,TAPEIO
8524 64C0;          1050         OUT TPESEL;,BTHDRV
8526 6101;          1051         OUT BDSEL;,TRMINL
8528 D4813E;        1052         SEP CALL;,A(READ)     ..WAIT FOR KEY
852B C083FF;        1053         LBR TPOFF
852E ;              1054
852E D48504;        1055 TPWIND: SEP CALL;,A(REWIND)  ..FOR THE UTILITY CALL "J"
8531 C087F0;        1056         LBR RENTER
8534 ;              1057
8534 ;              1058 ..*************************************************
8534 ;              1059 ..      STARTS A USER PROGRAM WITH SPECIFIED ADDRESS
8534 ;              1060 ..      IN REGISTER 0, AND X=0. THE SCREEN IS CLEARED
8534 ;              1061 ..      AND THE CURSOR IS HOMED.
8534 ;              1062 ..REG USED:  CHAR, ASL, R0
8534 ;              1063 ..*************************************************
8534 ;              1064
8534 D487EA;        1065 RUN:    SEP CALL;,A(READHX)      ..LOOK FOR STARTING ADDRESS
8537 FB0D;          1066         XRI CR                   ..FIRST NON-HEX MUST BE A
8539 CA8085;        1067         LBNZ ERROR               ..CR, ELSE SYNTAX ERROR
853C ;              1068
853C 9DB0;          1069 RUN1:   GHI ASL;PHI R0           ..GET THE ADDRESS
853E 8DA0;          1070         GLO ASL;PLO R0
8540 E0;            1071         SEX R0
8541 D0;            1072         SEP R0                   ..AND GO!
```

```
8542 ;              1073
8542 ;              1074 ..*****************************************************************
8542 ;              1075 ..                      READ TAPE ROUTINE
8542 ;              1076 ..      INITIALLY READS ONE BLOCK OF 384 BYTES INTO A BUFFER FROM TAPE
8542 ;              1077 ..      THEN EACH CALL TO IT PUTS ONE BYTE INTO CHAR.1 AND DECREMENTS
8542 ;              1078 ..      DOWN THE BUFFER UNTILL IT IS EMPTY THEN GETS NEXT BUFFER LOAD
8542 ;              1079 ..      USES A 3 BYTE BUFFER WHICH WHEN NOT IN USE IS STORED ON STACK
8542 ;              1080 ..              LOWEST ADDRESS IS BLOCK COUNT
8542 ;              1081 ..              NEXT IS DRIVE # IN TOP2 BITS AND MSB OF BYTE COUNT
8542 ;              1082 ..              HIGHEST IS REST OF BYTE COUNT
8542 ;              1083 ..      PTR (REGISTER C) NORMALLY POINTS AT THE HIGHEST BYTE
8542 ;              1084 ..  %%% NOTE: REGISTER C MUST CONTAIN THE IOCB ADDRESS AS SET UP
8542 ;              1085 ..            IN THE ROUTINE FINDTP.  %%%
8542 ;              1086 ..*****************************************************************
8542 ;              1087
8542 9373;          1088 READT:  GHI PC;STXD             ..INIT PARITY FLAG
8544 0C;            1089         LDN PTR                 ..IF BYTE CNT GREATER THAN 0, THEN JUST PUT IT
8545 3A7B;          1090         BNZ EXRD                ..IN CHAR.1
8547 2C;            1091         DEC PTR
8548 0CF6;          1092         LDN PTR;SHR             ..TEST HI BIT OF IOCB
854A 3379;          1093         BDF EXRD1
854C 0CF901;        1094         LDN PTR;ORI 1           ..SET BYTE COUNT TO MAXIMUM
854F 5C;            1095         STR PTR
8550 1C;            1096         INC PTR
8551 F8805C;        1097         LDI #80;STR PTR
8554 D48676;        1098         SEP CALL;,A(RDBLOK)     ..GET NEXT BLOCK FROM TAPE
8557 3B7A;          1099         BNF TAG1                ..IF NO PARITY ERR
8559 12;            1100         INC SP                  ..OTHERWISE SET PARITY FLAG
855A F80073;        1101         LDI 0;STXD
855D 9C73;          1102         GHI PTR;STXD            ..SAVE IOCB!
855F 8C73;          1103         GLO PTR;STXD
8561 D483F0;        1104         SEP CALL;,A(OSTRNG)
8564 0D0A5041524954;1105         ,A(CRLF),T'PARITY BAD'
856B 5920424144;    1105
8570 0D0A00;        1106         ,A(CRLF),0
8573 12;            1107         INC SP                  ..RESTORE IOCB
8574 42AC;          1108         LDA SP;PLO PTR
8576 02BC;          1109         LDN SP;PHI PTR
8578 38;            1110         SKP
8579 ;              1111
8579 1C;            1112 EXRD1:  INC PTR
857A ;              1113
857A 0C;            1114 TAG1:   LDN PTR                 ..DEC BYTE COUNT IN THE IOCB
857B FF015C;        1115 EXRD:   SMI  1;STR PTR
857E AF;            1116         PLO CHAR                ..CHAR POINTS INTO THE READ BUFFER
857F 2C;            1117         DEC PTR
8580 0C7F005C;      1118         LDN PTR;SMBI 0;STR PTR
8584 FA01FC8DBF;    1119         ANI 1;ADI A.1(RDBUFF-256);PHI CHAR
8589 1C;            1120         INC PTR
858A 0FBF;          1121         LDN CHAR;PHI CHAR       ..PUT THE BYTE INTO CHAR.1
858C 12;            1122         INC SP                  ..SET PARITY FLAG (1=PASS)
858D 02FE;          1123         LDN SP;SHL
858F D5;            1124         SEP R5
8590 ;              1125
8590 ;              1126 ..*****************************************************************
8590 ;              1127 ..      WRITE TAPE ROUTINE
8590 ;              1128 ..      PUTS BYTE FROM CHAR.1 INTO BLOCK BUFFER.
8590 ;              1129 ..      WHEN BUFFER FULL,WRITES IT TO SELECTED TAPE.
8590 ;              1130 ..       NOTE: REGISTER C MUST CONTAIN THE IOCB ADDRESS
8590 ;              1131 ..*****************************************************************
```

```
8590 ;                1132
8590 9F52;            1133 TWRITE: GHI CHAR;STR SP          ..SAVE BYTE
8592 0CFC80AF;        1134         LDN PTR;ADI A.0(WRTBUF);PLO CHAR
8596 2C;              1135         DEC PTR                  ..CHAR POINTS TO NEXT FREE
8597 0CFA01;          1136         LDN PTR;ANI 1            ..BUFFER LOCATION
859A 7C8EBF;          1137         ADCI A.1(WRTBUF);PHI CHAR
859D 025F;            1138         LDN SP;STR CHAR          ..WRITE BYTE TO BUFFER
859F 1F;              1139         INC CHAR
85A0 1C;              1140         INC PTR                  ..PT TO IOCB3
85A1 0CFC015C;        1141         LDN PTR;ADI 1;STR PTR    ..INC BYTE CNT
85A5 2C;              1142         DEC PTR
85A6 0C7C005C;        1143         LDN PTR;ADCI 0;STR PTR
85AA 1C;              1144         INC PTR
85AB 9FFB90;          1145         GHI CHAR;XRI #90         ..BUFF FULL?
85AE 3AC2;            1146         BNZ EXWT
85B0 D485FE;          1147         SEP CALL;,A(WRBLOK)      ..ELSE,WRITE BUFFER TO TAPE
85B3 2C2C;            1148         DEC PTR;DEC PTR          ..PT TO BLK CNT
85B5 0CFC015C;        1149         LDN PTR;ADI 1;STR PTR    ..INC CNT
85B9 1C;              1150         INC PTR                  ..POINT TO BYTE COUNT
85BA 0CFAFE;          1151         LDN PTR;ANI #FE          ..SET IT TO 0
85BD 5C1C;            1152         STR PTR;INC PTR
85BF F8005C;          1153         LDI 0;STR PTR
85C2 ;                1154
85C2 FF00;            1155 EXWT:   SMI 0
85C4 D5;              1156         SEP R5
85C5 ;                1157
85C5 ;                1158 ..*****************************************************************
85C5 ;                1159 ..      MESSAGE FOR NO RAM AT A LOCATION FOR STORE OF A READ
85C5 ;                1160 ..*****************************************************************
85C5 ;                1161
85C5 9C73;            1162 NOTRAM: GHI RC;STXD
85C7 8C73;            1163         GLO RC;STXD
85C9 D483F0;          1164         SEP CALL;,A(OSTRNG)
85CC 0D0A4E4F205241;1165           ,A(CRLF),T'NO RAM @',0
85D3 4D204000;        1165
85D7 98BF;            1166         GHI ADRPTR;PHI CHAR      ..TELL HIM THE ADDRESS
85D9 D481AE;          1167         SEP CALL;,A(TYPE2)
85DC 88BF;            1168         GLO ADRPTR;PHI CHAR
85DE D481AE;          1169         SEP CALL;,A(TYPE2)
85E1 12;              1170         INC SP
85E2 72AC;            1171         LDXA;PLO RC
85E4 02BC;            1172         LDN SP;PHI RC
85E6 D5;              1173         SEP R5
85E7 ;                1174 ..*****************************************************************
85E7 ;                1175 ..              ROM ENTRY FROM UTILITY
85E7 ;                1176 ..*****************************************************************
85E7 F890B0;          1177 EDITOR: LDI AEDIT;PHI R0
85EA 30F9;            1178         BR GOADD
85EC ;                1179
85EC F89BB0;          1180 ASSEMB: LDI AASM;PHI R0
85EF 30F9;            1181         BR GOADD
85F1 ;                1182
85F1 F8B0B0;          1183 BASIC:  LDI ABAS;PHI R0
85F4 30F9;            1184         BR GOADD
85F6 ;                1185
85F6 F8E0B0;          1186 PROMIT: LDI PRMPGR;PHI R0
85F9 ;                1187
85F9 F800A0;          1188 GOADD:  LDI  0;PLO R0
85FC E0;              1189         SEX R0
85FD D0;              1190         SEP R0                   ..GO TO ROM
```

```
85FE ;              1191
85FE ;              1192 ..*****************************************************************
85FE ;              1193 ..              WRITE BLOCK ROUTINE
85FE ;              1194 ..      WRITES A BLOCK OF 384 BYTES TO TAPE
85FE ;              1195 ..*****************************************************************
85FE ;              1196
85FE D48407;        1197 WRBLOK: SEP CALL;,A(REGSAV)
8601 8B52;          1198         GLO RB;STR SP              ..NEED EXTRA STORAGE
8603 F880AA;        1199         LDI A.0(WRTBUF);PLO BUFPTR..POINT TO WRITE BUFFER
8606 F88EBA;        1200         LDI A.1(WRTBUF);PHI BUFPTR
8609 AB;            1201         PLO BLKFLG                 ..INIT BLOCK FLAG
860A F859A8;        1202         LDI A.0(WRBIT);PLO BITWRT
860D F886B8;        1203         LDI A.1(WRBIT);PHI BITWRT
8610 E3;            1204         SEX PC
8611 6102;          1205         OUT BDSEL;,TAPEIO
8613 EC2C;          1206         SEX PTR;DEC PTR          ..POINT TO DRIVE #
8615 64;            1207         OUT TPESEL               ..SELECT DRIVE(S)
8616 2C2C;          1208         DEC PTR;DEC PTR          ..PT TO BLOCK COUNT
8618 0C;            1209         LDN PTR
8619 3A1E;          1210         BNZ LDR2
861B F8C0C8;        1211         LDI #C0;LSKP             ..COUNT FOR INITIAL LEADER
861E F810;          1212 LDR2:   LDI #10                  ..COUNT FOR INTER BLOCK LEADER
8620 B9;            1213         PHI LDRCNT
8621 FC00;          1214 LEADER: ADI 0                    ..SET DF=0 TO WRITE A LOW
8623 D8;            1215         SEP BITWRT               ..WRITE ONE ZERO
8624 29;            1216         DEC LDRCNT
8625 99;            1217         GHI LDRCNT
8626 3A21;          1218         BNZ LEADER
8628 F810A7;        1219 SET1:   LDI #10;PLO PARITY       ..PRESET PARITY EVEN
862B F808A9;        1220         LDI 8;PLO BITCNT         ..PRESET BIT COUNTER
862E 8B;            1221         GLO BLKFLG
862F 3237;          1222         BZ NOTBLK
8631 4CB7;          1223         LDA PTR;PHI BYTSTR       ..STORE THE BYTE TO BE WRITTEN
8633 F800AB;        1224         LDI 0;PLO BLKFLG         ..CLEAR THE BLOCK FLAG
8636 C8;            1225         LSKP                     ..IT IS THE BLOCK START
8637 ;              1226
8637 4AB7;          1227 NOTBLK: LDA BUFPTR;PHI BYTSTR    ..STORE FIRST BYTE
8639 FF00;          1228         SMI 0
863B D8;            1229         SEP BITWRT               ..WRITE START BIT
863C ;              1230
863C 97F6;          1231 SET:    GHI BYTSTR;SHR           ..MOVE EACH BIT INTO DF FOR
863E B7;            1232         PHI BYTSTR               ..OUTPUT
863F D8;            1233         SEP BITWRT
8640 29;            1234         DEC BITCNT
8641 89;            1235         GLO BITCNT
8642 3A3C;          1236         BNZ SET                  ..DO IT FOR ALL BITS
8644 17;            1237         INC PARITY               ..SET ODD PARITY
8645 87F6;          1238         GLO PARITY;SHR
8647 D8;            1239         SEP BITWRT               ..WRITE OUT PARITY
8648 9AFB90;        1240         GHI BUFPTR;XRI #90       ..BUFFER DONE?
864B 3A28;          1241         BNZ SET1
864D D8;            1242         SEP BITWRT               ..REPEAT PARITY FOR 2 BITS
864E D8;            1243         SEP BITWRT
864F 1C;            1244         INC PTR                  ..PT TO IOCB#3
8650 02AB;          1245         LDN SP;PLO RB            ..REPLACE REG B
8652 ;              1246
8652 D487D7;        1247 RESTOR: SEP CALL;,A(REGSTR)
8655 C083FF;        1248         LBR TPOFF
8658 ;              1249 ..*****************************************************************
8658 ;              1250 ..              ROUTINE TO WRITE A BIT ONTO TAPE
```

```
8658 ;              1251 ..              ENTERED WITH SEP TO BITWRTR
8658 ;              1252 ..*****************************************************************
8658 ;              1253
8658 D3;            1254 BYE:    SEP PC
8659 ;              1255
8659 E8;            1256 WRBIT:  SEX BITWRT
865A F8FFAF;        1257         LDI #FF;PLO FLAG        ..FLAG FOR SIGNAL HIGH (FF),
865D ;              1258                                 ..LOW (00)
865D F80C;          1259         LDI #0C                 ..0 BIT TIMING
865F 3B64;          1260         BNF OUTAPE
8661 F827;          1261         LDI #27                 ..1 BIT TIMING
8663 17;            1262         INC PARITY              ..INC PARITY CTR IF 1 BIT
8664 ;              1263
8664 6501;          1264 OUTAPE: OUT TPEWRT;,HIGH
8666 BF;            1265         PHI CHAR                ..SAVE BIT TIMING IN CHAR.1
8667 FF01;          1266 TEST1:  SMI 1                   ..DEC TIMER
8669 3A67;          1267         BNZ TEST1               ..KEEP HI TIL CNT = 0
866B 8F;            1268         GLO FLAG
866C 3258;          1269         BZ BYE          ..EXIT WHEN TAPE SIGNAL IS A LOW
866E F800AF;        1270         LDI 0;PLO FLAG  ..SET FLAG FOR A LOW
8671 6500;          1271         OUT TPEWRT;,LOW
8673 9F;            1272         GHI CHAR        ..GET THE BIT TIMING
8674 3067;          1273         BR TEST1
8676 ;              1274
8676 ;              1275 ..*****************************************************************
8676 ;              1276 ..                      READ BLOCK ROUTINE
8676 ;              1277 ..      READS ONE BLOCK, 384 BYTES, FROM TAPE INTO THE READ BUFFER
8676 ;              1278 ..      EXITS WITH DF AS PARITY FLAG, 0 IS GOOD
8676 ;              1279 ..*****************************************************************
8676 ;              1280
8676 D48407;        1281 RDBLOK: SEP CALL;,A(REGSAV)
8679 F87FAA;        1282         LDI A.0(RDBUFF);PLO BUFPTR
867C F88EBA;        1283         LDI A.1(RDBUFF);PHI BUFPTR
867F BF;            1284         PHI FLAG                ..INITIALIZE BLOCK FLAG TO SET
8680 F800AF;        1285         LDI 0;PLO FLAG          ..AND PARITY FLAG TO GOOD
8683 F81DA8;        1286         LDI A.0(RDBIT);PLO BITRDR
8686 F883B8;        1287         LDI A.1(RDBIT);PHI BITRDR
8689 E3;            1288         SEX PC
868A 6102;          1289         OUT BDSEL;,TAPEIO
868C 2C;            1290         DEC PTR                 ..MOVE TO DRIVE # IN IOCB
868D 0CFE;          1291         LDN PTR;SHL
868F 1C;            1292         INC PTR                 ..BACK TO HOME IN IOCB
8690 3395;          1293         BDF DR1
8692 6440;          1294         OUT TPESEL;,DRIVE0
8694 C8;            1295         LSKP
8695 ;              1296
8695 6480;          1297 DR1:    OUT TPESEL;,DRIVE1
8697 ;              1298
8697 F808B9;        1299 LDR1:   LDI #08;PHI LDRCNT
869A ;              1300
869A D8;            1301 LEDER1: SEP BITRDR              ..GET 1 BIT
869B 3397;          1302         BDF LDR1          ..MUST BE A 0 TO BE THE LEADER
869D 29;            1303         DEC LDRCNT
869E 99;            1304         GHI LDRCNT
869F 3A9A;          1305         BNZ LEDER1        ..LOOP TIL LEADER COUNTER EMPTYS
86A1 ;              1306
86A1 D8;            1307 STRT1:  SEP BITRDR        ..LOOP TIL START BIT
86A2 3BA1;          1308         BNF STRT1
86A4 F809A9;        1309         LDI 9;PLO BITCNT
86A7 A7;            1310         PLO PARITY
```

```
86A8 ;                1311
86A8 97;              1312 SHIFT:  GHI TMPRG1        ..KEEP SHIFTING THE BIT INTO THE MSB
86A9 76B7;            1313         SHRC;PHI TMPRG1   ..POSITION
86AB 29;              1314         DEC BITCNT
86AC D8;              1315         SEP BITRDR
86AD 89;              1316         GLO BITCNT
86AE 3AA8;            1317         BNZ SHIFT         ..ASSEMBLE THE BYTE
86B0 87;              1318         GLO PARITY        ..READ PARITY ON LAST PASS
86B1 F6;              1319         SHR
86B2 CF;              1320         LSDF              ..FOR NO PARITY ERROR
86B3 93AF;            1321         GHI PC;PLO FLAG   ..ELSE SET PARITY FLAG
86B5 9F;              1322         GHI FLAG          ..IS IT BLOCK # ?
86B6 32C3;            1323         BZ BYTE
86B8 2C2C;            1324         DEC PTR;DEC PTR   ..POINT TO BLOCK #
86BA 975C;            1325         GHI TMPRG1;STR PTR..WRITE IT TO IOCB
86BC 1C1C;            1326         INC PTR;INC PTR   ..POINT TO HOME
86BE F800BF;          1327         LDI 0;PHI FLAG    ..RESET BLK FLAG
86C1 30A1;            1328         BR STRT1
86C3 ;                1329
86C3 975A;            1330 BYTE:   GHI TMPRG1;STR BUFPTR ..PUT BYTE IN BUFFER
86C5 2A;              1331         DEC BUFPTR
86C6 8AFBFF;          1332         GLO BUFPTR;XRI #FF
86C9 3AA1;            1333         BNZ STRT1
86CB 9AFB8C;          1334         GHI BUFPTR;XRI #8C
86CE 3AA1;            1335         BNZ STRT1         ..UNTIL BUFFER IS FULL
86D0 8FFE;            1336         GLO FLAG;SHL              ..SET/RESET PARITY
86D2 3052;            1337         BR RESTOR
86D4 ;                1338
86D4 ;                1339 ..*****************************************************************************
86D4 ;                1340 ..                        TAPE SAVE ROUTINE
86D4 ;                1341 ..                 COPIES DATA FROM MEMORY TO TAPE
86D4 ;                1342 ..*****************************************************************************
86D4 ;                1343
86D4 D483F0;          1344 TPWRIT: SEP CALL;,A(OSTRNG)
86D7 0D0A46524F4D20;1345           ,A(CRLF),T'FROM ',0
86DE 00;              1345
86DF F800B7A7;        1346         LDI 0;PHI SRTADD;PLO SRTADD..CLEAR THE START ADDRESS
86E3 A9;              1347         PLO TMPRG3            ..POINT TO BEGINNING OF THE BUFFER
86E4 BDAD;            1348         PHI ASL;PLO ASL
86E6 D48314;          1349         SEP CALL;,A(READCR)
86E9 9DB7;            1350         GHI ASL;PHI SRTADD
86EB 8DA7;            1351         GLO ASL;PLO SRTADD
86ED F800ADBD;        1352         LDI 0;PLO ASL;PHI ASL
86F1 D483F0;          1353         SEP CALL;,A(OSTRNG)
86F4 0A544F2000;      1354         ,LF,T'TO ',0
86F9 D48314;          1355         SEP CALL;,A(READCR)
86FC 9DB8;            1356         GHI ASL;PHI ENDADD
86FE 8DA8;            1357         GLO ASL;PLO ENDADD
8700 18;              1358         INC ENDADD
8701 D48774;          1359         SEP CALL;,A(FINDTP)
8704 D487A6;          1360         SEP CALL;,A(WRITEX)   ..WRITE AN "I" TO TAPE FOR
8707 490D0A00;        1361         ,T'I',A(CRLF),0       .."INSERT"
870B D48752;          1362         SEP CALL;,A(WRTH)     ..WRITE THE ADDRESS
870E 8852;            1363 CKEND:  GLO ENDADD;STR SP     ..DID WE DO ALL THE ADDRESSES?
8710 87F3;            1364         GLO SRTADD;XOR
8712 3A1A;            1365         BNZ WRTD2
8714 9852;            1366         GHI ENDADD;STR SP
8716 97F3;            1367         GHI SRTADD;XOR
8718 3230;            1368         BZ FINISH
871A ;                1369
```

```
871A 87FA1F;          1370 WRTD2:  GLO SRTADD;ANI #1F ..CHECK IF 32 PAIRS OF DATA ARE
871D 3A29;            1371         BNZ WRTD1          ..PRINTED IN ONE LINE
871F D487A6;          1372         SEP CALL;,A(WRITEX)
8722 3B0D0A00;        1373         ,T';',A(CRLF),0    ..";" ENDS THE HEX DATA ON A LINE
8726 D48752;          1374         SEP CALL;,A(WRTH)  ..WRITE THE NEXT ADDRESS
8729 ;                1375
8729 47B9;            1376 WRTD1:  LDA SRTADD;PHI TMPRG3..STORE THE DATA BYTE
872B D487B7;          1377         SEP CALL;,A(ASCII)    ..CONVERT THE BYTE TO ASCII
872E ;                1378                               ..AND WRITE IT OUT
872E 300E;            1379         BR CKEND
8730 ;                1380
8730 D487A6;          1381 FINISH: SEP CALL;,A(WRITEX)   ..AT END WRITE OUT
8733 0D1300;          1382         ,CR,EOF,0             ..RETURN AND'DC3'
8736 F880A9;          1383         LDI #80;PLO TMPRG3    ..LOAD A NUMBER SO THAT WE CAN
8739 F8FEB9;          1384         LDI #FE;PHI TMPRG3    ..FILL THE BUFFER WITH 383 BYTES
873C ;                1385
873C F813BF;          1386 FILBFR: LDI EOF;PHI CHAR      ..OF DC3'S
873F D48590;          1387         SEP CALL;,A(TWRITE)
8742 1999;            1388         INC TMPRG3;GHI TMPRG3
8744 3A3C;            1389         BNZ FILBFR
8746 D483F0;          1390         SEP CALL;,A(OSTRNG)
8749 0A444F4E4500;    1391         ,LF,T'DONE',0
874F ;                1392
874F C087F0;          1393         LBR RENTER
8752 ;                1394
8752 ;                1395 ..*****************************************************************************
8752 ;                1396                 ..ROUTINE TO WRITE A HEX  ADDRESS TO TAPE
8752 ;                1397 ..*****************************************************************************
8752 ;                1398
8752 97B9;            1399 WRTH:   GHI SRTADD;PHI TMPRG3
8754 D487B7;          1400         SEP CALL;,A(ASCII)
8757 87B9;            1401         GLO SRTADD;PHI TMPRG3
8759 D487B7;          1402         SEP CALL;,A(ASCII)
875C D487A6;          1403         SEP CALL;,A(WRITEX)
875F 2000;            1404         ,SPACE,0
8761 ;                1405
8761 D5;              1406 WRITER: SEP R5
8762 ;                1407
8762 ;                1408 ..*************************************************************
8762 ;                1409 ..              ROUTINE TO GET THE TAPE DRIVE NUMBER
8762 ;                1410 ..              AND SET UP THE IOCB
8762 ;                1411 ..*************************************************************
8762 ;                1412
8762 ;                1413                 ORG #8774  ..TO COMPENSATE FOR THE PROM PROGRAMMER
8774 ;                1414
8774 D483F0;          1415 FINDTP: SEP CALL;,A(OSTRNG)
8777 0D0A5441504523;1416         ,A(CRLF),T'TAPE# ',0
877E 2000;            1416
8780 ;                1417
8780 F800AD;          1418 ASK2:   LDI 0;PLO ASL
8783 D48314;          1419         SEP CALL;,A(READCR)       ..GET THE DRIVE #
8786 8DF6;            1420         GLO ASL;SHR               ..VALID ENTRY?
8788 3298;            1421         BZ FOUND
878A D483F0;          1422         SEP CALL;,A(OSTRNG)
878D 0A30204F522031;1423         ,LF,T'0 OR 1?',0
8794 3F00;            1423
8796 3080;            1424         BR ASK2
8798 ;                1425
8798 ;                1426                 ..FORM THE IOCB VALUES ON THE STACK
8798 ;                1427
```

```
8798 73;           1428 FOUND:  STXD                    ..STORE IT FOR IOCB
8799 F840;         1429         LDI DRIVE0
879B 3B9E;         1430         BNF DRBIT               ..IF DRIVE 0 SELECTED
879D FE;           1431         SHL                     ..ELSE SET TO 1
879E ;             1432
879E 73;           1433 DRBIT:  STXD                    ..STORE 0 COUNT + DRIVE FOR IOCB
879F F80073;       1434         LDI 0;STXD              ..SET BLOCK#=0
87A2 30AF;         1435         BR FIXERR               ..NEED TO JUMP OVER THE NEXT ORG
87A4 ;             1436
87A4 ;             1437 ..*************************************************************
87A4 ;             1438 ..              TAPE WRITE SUBROUTINES
87A4 ;             1439 ..*************************************************************
87A4 ;             1440
87A4 ;             1441                 ..WRITE IMMEDIATE
87A4 ;             1442
87A4 ;             1443                 ORG #87A6       ..ONCE AGAIN FOR THE PROM PROGRAMMER
87A6 ;             1444
87A6 46BF;         1445 WRITEX: LDA  LINK;PHI CHAR      ..GET THE IMMEDIATE BYTE AND
87A8 3261;         1446         BZ WRITER
87AA D48590;       1447         SEP CALL;,A(TWRITE)     ..WRITE IT TO TAPE UNTIL GET
87AD 30A6;         1448         BR WRITEX               .. A 0
87AF ;             1449
87AF ;             1450 ..*************************************************************
87AF ;             1451 ..              REST OF THE FIND TAPE ROUTINE
87AF ;             1452 ..*************************************************************
87AF ;             1453
87AF 92BC;         1454 FIXERR: GHI SP;PHI PTR          ..POINT PTR TO IOCB AREA
87B1 82AC;         1455         GLO SP;PLO PTR
87B3 1C1C;         1456         INC PTR;INC PTR
87B5 1C;           1457         INC PTR                 ..POINT TO HOME IN IOCB
87B6 D5;           1458         SEP R5
87B7 ;             1459
87B7 ;             1460 ..*************************************************************
87B7 ;             1461                 ..ROUTINE TO OUTPUT ASCII BYTE TO TAPE
87B7 ;             1462 ..*************************************************************
87B7 ;             1463
87B7 99;           1464 ASCII:  GHI TMPRG3                ..DO THE HIGH NIBBLE
87B8 F6F6F6F6A9;   1465         SHR;SHR;SHR;SHR;PLO TMPRG3
87BD D487C8;       1466         SEP CALL;,A(ASCIIR)
87C0 99FA0F;       1467         GHI TMPRG3;ANI #0F      ..AND THE LOW ONE
87C3 A9;           1468         PLO TMPRG3
87C4 D487C8;       1469         SEP CALL;,A(ASCIIR)
87C7 D5;           1470         SEP R5
87C8 ;             1471
87C8 89FC30BF;     1472 ASCIIR: GLO TMPRG3;ADI #30;PHI CHAR
87CC FF3A3BD3;     1473         SMI #3A;BM ASC1 ..FOR 0 TO 9
87D0 FC41BF;       1474         ADI #41;PHI CHAR..FOR A TO F
87D3 ;             1475
87D3 D48590;       1476 ASC1:   SEP CALL;,A(TWRITE)
87D6 D5;           1477         SEP R5
87D7 ;             1478
87D7 ;             1479 ..*************************************************************
87D7 ;             1480 ..                  REGISTER RESTORE
87D7 ;             1481 ..        RESTORES THE REGISTERS SAVED BY REGSAV,
87D7 ;             1482 ..        ASSUMES STACK POINTER IS 1 BYTE ABOVE THE
87D7 ;             1483 ..        STORED DATA. EXITS POINTING TO A FREE BYTE
87D7 ;             1484 ..*************************************************************
87D7 ;             1485
87D7 60;           1486 REGSTR: IRX
87D8 72B7;         1487         LDXA;PHI TMPRG1
```

```
87DA 72A7;          1488         LDXA;PLO TMPRG1
87DC 72B8;          1489         LDXA;PHI TMPRG2
87DE 72A8;          1490         LDXA;PLO TMPRG2
87E0 72B9;          1491         LDXA;PHI TMPRG3
87E2 72A9;          1492         LDXA;PLO TMPRG3
87E4 72BA;          1493         LDXA;PHI CNT
87E6 02AA;          1494         LDN SP;PLO CNT
87E8 9F;            1495         GHI CHAR            ..TO COMPENSATE FOR BASIC
87E9 D5;            1496         SEP R5
87EA ;              1497
87EA ;              1498 ..*************************************************
87EA ;              1499 ..      FILLS ASL AS LONG AS HEX DIGITS ARE ENTERED
87EA ;              1500 ..*************************************************
87EA ;              1501
87EA D4813B;        1502 READHX: SEP CALL;,A(READAH)
87ED 33EA;          1503         BDF READHX
87EF D5;            1504         SEP R5
87F0 ;              1505
87F0 ;              1506 ..*********************************************************************
87F0 ;              1507 ..      ENTRY POINT FOR BASIC "BYE" COMMAND AND FOR GENERAL REENTER
87F0 ;              1508 ..*********************************************************************
87F0 ;              1509
87F0 ;              1510                 ORG #87F0
87F0 ;              1511
87F0 F8F7A0;        1512 RENTER: LDI A.0(RENTR1);PLO R0 ..CAN BE ENTERED WITH X
87F3 F887B0;        1513         LDI A.1(RENTR1);PHI R0 .. AND P SET TO
87F6 D0;            1514         SEP R0                 ..ANYTHING. RESETS ALL THE SCRT
87F7 F82FA3;        1515 RENTR1: LDI A.0(PRMPT);PLO PC   ..REGISTERES.
87FA F880B3;        1516         LDI A.1(PRMPT);PHI PC
87FD C0838F;        1517         LBR ENTER2
8800 ;              1518
8800 ;              1519
```

Appendix C —
Tape Handling Precautions

The tape recorders used in the MCDS have unshielded record heads which may be subject to noise pick up from magnetic interference.

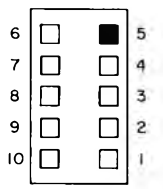
Magnetic fields can be caused by electrical equipment such as CRT displays, motors, solenoids and other industrial equipment. This interference will show up as PARITY BAD errors during tape read. If these errors occur, make sure the tape recorder is as far away from noise sources, especially CRT's, as possible. Metal-enclosed CRT's such as RCA model TC1110 are suggested. Under extreme conditions, it may be necessary to shield the recorders.

Pick-up from noisy power lines may also cause PARITY BAD errors during tape read. Running the tape recorders on batteries or with a 6-volt DC power supply to the DC input on the recorder should help.

Appendix D —
Connection List for Terminal Interface Cable CDP18S516

CDP18S516
EIA RS232C Terminal

P1	P2	Signal
1	1	Ground
2	2	Data to MCDS
3	3	Data to Terminal
10	7	Signal Ground
7	5	Clear to Send— Held High by MCDS
6	6,8	Data Set Ready— Held High by MCDS



Appendix E—
Instruction Summary for RCA CDP1802A, CDP1804, and CDP1805 COSMAC Microprocessors

The COSMAC instruction summary is given in the tabulations below. Hexadecimal notation is used to refer to the 4-bit binary codes.

In all registers bits are numbered from the least significant bit (LSB) to the most significant bit (MSB) starting with 0.

R(W): Register designated by W, where W= N or X, or P

R(W).0: Lower-order byte of R(W)
R(W).1: Higher-order byte of R(W)
N0= Least significant bit of N Register

Operation Notation
M(R(N))→D;R(N)+1

This notation means: The memory byte pointed to by R(N) is loaded into D, and R(N) is incremented by 1.

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
MEMORY REFERENCE			
LOAD VIA N	LDN	0N	M(R(N))·D; FOR N NOT 0
LOAD ADVANCE	LDA	4N	M(R(N))·D; R(N) +1 → R(N)
LOAD VIA X	LDX	F0	M(R(X))·D
LOAD VIA X AND ADVANCE	LDXA	72	M(R(X))·D; R(X) + 1 → R(X)
LOAD IMMEDIATE	LDI	F8	M(R(P))·D; R(P) +1 → R(P)
STORE VIA N	STR	5N	D·M(R(N))
STORE VIA X AND DECREMENT	STXD	73	D·M(R(X)); R(X) −1 → R(X)
REGISTER OPERATIONS			
INCREMENT REG N	INC	1N	R(N) +1 → R(N)
DECREMENT REG N	DEC	2N	R(N) −1 → R(N)
INCREMENT REG X	IRX	60	R(X) +1 → R(X)
GET LOW REG N	GLO	8N	R(N).0·D
PUT LOW REG N	PLO	AN	D·R(N).0
GET HIGH REG N	GHI	9N	R(N).1·D
PUT HIGH REG N	PHI	BN	D·R(N).1
LOGIC OPERATIONS◆◆			
OR	OR	F1	M(R(X)) OR D·D
OR IMMEDIATE	ORI	F9	M(R(P)) OR D→D; R(P) +1→R(P)
EXCLUSIVE OR	XOR	F3	M(R(X)) XOR D→D
EXCLUSIVE OR IMMEDIATE	XRI	FB	M(R(P)) XOR D→D; R(P) +1→R(P)
AND	AND	F2	M(R(X)) AND D→D
AND IMMEDIATE	ANI	FA	M(R(P)) AND D→D; R(P) +1→R(P)
SHIFT RIGHT	SHR	F6	SHIFT D RIGHT, LSB(D)→DF, 0→MSB(D)
SHIFT RIGHT WITH CARRY	SHRC	76◆	SHIFT D RIGHT, LSB(D)→DF, DF→MSB(D)
RING SHIFT RIGHT	RSHR		
SHIFT LEFT	SHL	FE	SHIFT D LEFT, MSB(D)→DF, 0→LSB(D)
SHIFT LEFT WITH CARRY	SHLC	7E◆	SHIFT D LEFT, MSB(D)→DF, DF→LSB(D)
RING SHIFT LEFT	RSHL		
REGISTER LOAD VIA X AND ADVANCE	RLXA *	686N	M(R(X))→R(N).1; M(R(X) +1)→ R(N).0; R(X) +2→R(X)
REGISTER LOAD IMMEDIATE	RLDI *	68CN	M(R(P))→R(N).1; M(R(P) +1)→ R(N).0; R(P) +2→R(P)
REGISTER STORE VIA X AND DECREMENT	RSXD *	68AN	R(N).0→M(R(X)); R(N).1→ M(R(X)-1); R(X) −2→R(X)
REGISTER N TO REGISTER X COPY	RNX *	68BN	R(N)→R(X)

For Notes, see page 90.

*CDP1804 and CDP1805 only.

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
ARITHMETIC OPERATIONS◆◆			
ADD	ADD	F4	M(R(X)) +D→DF, D
ADD IMMEDIATE	ADI	FC	M(R(P)) +D→DF, D; R(P) +1→R(P)
ADD WITH CARRY	ADC	74	M(R(X)) +D +DF→DF, D
ADD WITH CARRY IMMEDIATE	ADCI	7C	M(R(P)) +D +DF→DF, D R(P) +1→R(P)
SUBTRACT D	SD	F5	M(R(X))−D→DF, D
SUBTRACT D IMMEDIATE	SDI	FD	M(R(P))−D→DF, D; R(P) +1→R(P)
SUBTRACT D WITH BORROW	SDB	75	M(R(X))−D−(NOT DF)→DF, D
SUBTRACT D WITH BORROW, IMMEDIATE	SDBI	7D	M(R(P))−D−(NOT DF)→DF, D; R(P) +1→R(P)
SUBTRACT MEMORY	SM	F7	D−M(R(X))→DF, D
SUBTRACT MEMORY IMMEDIATE	SMI	FF	D−M(R(P))→DF, D; R(P) +1→R(P)
SUBTRACT MEMORY WITH BORROW	SMB	77	D−M(R(X))−(NOT DF)→DF, D
SUBTRACT MEMORY WITH BORROW, IMMEDIATE	SMBI	7F	D−M(R(P))−(NOT DF)→DF, D R(P) +1→R(P)
BRANCH INSTRUCTIONS−SHORT BRANCH			
SHORT BRANCH	BR	30	M(R(P))→R(P).0
NO SHORT BRANCH (SEE SKP)	NBR	38◆	R(P) +1→R(P)
SHORT BRANCH IF D=0	BZ	32	IF D=0, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH IF D NOT 0	BNZ	3A	IF D NOT 0, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH IF DF=1	BDF	33◆	IF DF=1, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH IF POS OR ZERO	BPZ		
SHORT BRANCH IF EQUAL OR GREATER	BGE		
SHORT BRANCH IF DF=0	BNF	3B◆	IF DF=0, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH IF MINUS	BM		
SHORT BRANCH IF LESS	BL		
SHORT BRANCH IF Q=1	BQ	31	IF Q=1, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH IF Q=0	BNQ	39	IF Q=0, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH IF EF1=1 ($\overline{EF1} = V_{SS}$)	B1	34	IF EF1=1, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH IF EF1=0 ($\overline{EF1} = V_{DD}$)	BN1	3C	IF EF1=0, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH IF EF2=1 ($\overline{EF2} = V_{SS}$)	B2	35	IF EF2=1, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH IF EF2=0 ($\overline{EF2} = V_{DD}$)	BN2	3D	IF EF2=0, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH IF EF3=1 ($\overline{EF3} = V_{SS}$)	B3	36	IF EF3=1, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH IF EF3=0 ($\overline{EF3} = V_{DD}$)	BN3	3E	IF EF3=0, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH IF EF4=1 ($\overline{EF4} = V_{SS}$)	B4	37	IF EF4=1, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH IF EF4=0 ($\overline{EF4} = V_{DD}$)	BN4	3F	IF EF4=0, M(R(P))→R(P).0 ELSE R(P) +1→R(P)
SHORT BRANCH ON COUNTER INTERRUPT	BCI *	683E	IF CI=1, M(R(P))→R(P).0, 0→CI ELSE R(P) +1→R(P)
SHORT BRANCH ON EXTERNAL INTERRUPT	BXI *	683F	IF XI=1, M(R(P))→R(P).0 ELSE R(P) +1→R(P)

◆NOTE THIS INSTRUCTION IS ASSOCIATED WITH MORE THAN ONE MNEMONIC EACH MNEMONIC IS INDIVIDUALLY LISTED

◆◆NOTE THE ARITHMETIC OPERATIONS AND THE SHIFT INSTRUCTIONS ARE THE ONLY INSTRUCTIONS THAT CAN ALTER THE DF AFTER AN ADD INSTRUCTION
DF = 1 DENOTES A CARRY HAS OCCURRED
DF = 0 DENOTES A CARRY HAS NOT OCCURRED
AFTER A SUBTRACT INSTRUCTION
DF = 1 DENOTES NO BORROW D IS A TRUE POSITIVE NUMBER
DF = 0 DENOTES A BORROW D IS TWO'S COMPLEMENT
THE SYNTAX "−(NOT DF)" DENOTES THE SUBTRACTION OF THE BORROW

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
BRANCH INSTRUCTIONS—LONG BRANCH			
LONG BRANCH	LBR	C0	M(R(P))→R(P).1 M(R(P) +1)→R(P).0
NO LONG BRANCH (SEE LSKP)	NLBR	C8♦	R(P) +2→R(P)
LONG BRANCH IF D=0	LBZ	C2	IF D=0, M(R(P))→R(P).1 M(R(P) +1)→R(P).0 ELSE R(P) +2→R(P)
LONG BRANCH IF D NOT 0	LBNZ	CA	IF D NOT 0, M(R(P))→R(P).1 M(R(P) +1)→R(P).0 ELSE R(P) +2→R(P)
LONG BRANCH IF DF=1	LBDF	C3	IF DF=1, M(R(P))→R(P).1 M(R(P) +1)→R(P).0 ELSE R(P) +2→R(P)
LONG BRANCH IF DF=0	LBNF	CB	IF DF=0, M(R(P))→R(P).1 M(R(P) +1)→R(P).0 ELSE R(P) +2→R(P)
LONG BRANCH IF Q=1	LBQ	C1	IF Q=1, M(R(P))→R(P).1 M(R(P) +1)→R(P).0 ELSE R(P) +2→R(P)
LONG BRANCH IF Q 0	LBNQ	C9	IF Q=0, M(R(P))→R(P).1 M(R(P) +1)→R(P).0 ELSE R(P) +2→R(P)
SKIP INSTRUCTIONS			
SHORT SKIP (SEE NBR)	SKP	38♦	R(P) +1→R(P)
LONG SKIP (SEE NLBR)	LSKP	C8♦	R(P) +2→R(P)
LONG SKIP IF D=0	LSZ	CE	IF D=0, R(P) +2→R(P) ELSE CONTINUE
LONG SKIP IF D NOT 0	LSNZ	C6	IF D NOT 0, R(P) +2→R(P) ELSE CONTINUE
LONG SKIP IF DF 1	LSDF	CF	IF DF=1, R(P) +2→R(P) ELSE CONTINUE
LONG SKIP IF DF=0	LSNF	C7	IF DF=0, R(P) +2→R(P) ELSE CONTINUE
LONG SKIP IF Q 1	LSQ	CD	IF Q=1, R(P) +2→R(P) ELSE CONTINUE
LONG SKIP IF Q 0	LSNQ	C5	IF Q=0, R(P) +2→R(P) ELSE CONTINUE
LONG SKIP IF IE 1	LSIE	CC	IF IE=1, R(P) +2→R(P) ELSE CONTINUE
CONTROL INSTRUCTIONS			
IDLE	IDL	00=	WAIT FOR DMA OR INTERRUPT; M(R(0))→BUS
NO OPERATION	NOP	C4	CONTINUE
SET P	SEP	DN	N→P
SET X	SEX	EN	N→X
SET Q	SEQ	7B	1→Q
RESET Q	REQ	7A	0→Q
PUSH X,P TO STACK	MARK	79	(X,P)→T; (X,P)→M(R(2)) THEN P→X; R(2)−1→R(2)
ENABLE TOGGLE Q	ETQ *	6809■	COUNTER = 01 AND COUNTER CLOCK ◄ ⇒ $\overline{Q}$→Q

■ETQ cleared by LDC, reset of CPU, or BCI • (CI = 1).

=An idle instruction initiates a repeating S1 cycle. The processor will continue to idle until an I/O request ($\overline{\text{INTERRUPT}}$, $\overline{\text{DMA-IN}}$, or $\overline{\text{DMA-OUT}}$) is activated. When the request is acknowledged, the IDLE cycle is terminated and the I/O request is serviced, and then normal operation is resumed.

♦NOTE THIS INSTRUCTION IS ASSOCIATED WITH MORE THAN ONE MNEMONIC. EACH MNEMONIC IS INDIVIDUALLY LISTED

*CDP1804 and CDP1805 only.

INSTRUCTION	MNEMONIC	OP CODE	OPERATION
COUNTER INSTRUCTIONS			
LOAD COUNTER	LDC *	6806	D→COUNTER
GET COUNTER	GEC *	6808	COUNTER→D
STOP COUNTER	STPC *	6800	0→COUNTER CLOCK
DECREMENT COUNTER	DTC *	6801	COUNTER − 1→COUNTER
SET TIMER MODE AND START	STM *	6807	TPA÷32→COUNTER CLOCK
SET COUNTER MODE 1 AND START	SCM1 *	6805	EF1→COUNTER CLOCK
SET COUNTER MODE 2 AND START	SCM2 *	6803	EF2→COUNTER CLOCK
SET PULSE WIDTH MODE 1, START	SPM1 *	6804	TPA·EF1→COUNTER CLOCK; $\overline{EF1}$ ↓ STOPS COUNT
SET PULSE WIDTH MODE 2, START	SPM2 *	6802	TPA·EF2→COUNTER CLOCK; $\overline{EF2}$ ↓ STOPS COUNT
INTERRUPT CONTROL			
EXTERNAL INTERRUPT ENABLE	XIE	680A	1→XIE
EXTERNAL INTERRUPT DISABLE	XID	680B	0→XIE
COUNTER INTERRUPT ENABLE	CIE	680C	1→CIE
COUNTER INTERRUPT DISABLE	CID	680D	0→CIE
RETURN	RET	70	M(R(X))→X,P; R(X)+1→R(X);1→IE
DISABLE	DIS	71	M(R(X))→X,P; R(X)+1→R(X);0→IE
SAVE	SAV	78	T→M(R(X))
INPUT–OUTPUT BYTE TRANSFER			
OUTPUT 1	OUT 1	61	M(R(X))→BUS;R(X)+1→R(X);N LINES=1
OUTPUT 2	OUT 2	62	M(R(X))→BUS;R(X)+1→R(X);N LINES=2
OUTPUT 3	OUT 3	63	M(R(X))→BUS;R(X)+1→R(X);N LINES=3
OUTPUT 4	OUT 4	64	M(R(X))→BUS;R(X)+1→R(X);N LINES=4
OUTPUT 5	OUT 5	65	M(R(X))→BUS;R(X)+1→R(X);N LINES=5
OUTPUT 6	OUT 6	66	M(R(X))→BUS;R(X)+1→R(X);N LINES=6
OUTPUT 7	OUT 7	67	M(R(X))→BUS;R(X)+1→R(X);N LINES=7
INPUT 1	INP 1	69	BUS→M(R(X)); BUS→D; N LINES = 1
INPUT 2	INP 2	6A	BUS→M(R(X)); BUS→D; N LINES = 2
INPUT 3	INP 3	6B	BUS→M(R(X)); BUS→D; N LINES = 3
INPUT 4	INP 4	6C	BUS→M(R(X)); BUS→D; N LINES = 4
INPUT 5	INP 5	6D	BUS→M(R(X)); BUS→D; N LINES = 5
INPUT 6	INP 6	6E	BUS→M(R(X)); BUS→D; N LINES = 6
INPUT 7	INP 7	6F	BUS→M(R(X)); BUS→D; N LINES = 7
CALL AND RETURN			
STANDARD CALL	SCAL *	688N	R(N).0→M(R(X)); R(N).1→M(R(X)−1); R(X)−2→R(X); R(P)→ R(N); THEN M(R(N))→ R(P).1; M(R(N)+1)→R(P).0; R(N)+2→R(N)
STANDARD RETURN	SRET *	689N	R(N)→R(P); M(R(X)+1) → R(N).1; M(R(X)+2)→ R(N).0; R(X)+2→R(X)

♦NOTE THIS INSTRUCTION IS ASSOCIATED WITH MORE THAN ONE MNEMONIC. EACH MNEMONIC IS INDIVIDUALLY LISTED

♦♦NOTE THE ARITHMETIC OPERATIONS AND THE SHIFT INSTRUCTIONS ARE THE ONLY INSTRUCTIONS THAT CAN ALTER THE DF AFTER AN ADD INSTRUCTION:
- DF = 1 DENOTES A CARRY HAS OCCURRED
- DF = 0 DENOTES A CARRY HAS NOT OCCURRED

AFTER A SUBTRACT INSTRUCTION:
- DF = 1 DENOTES NO BORROW. D IS A TRUE POSITIVE NUMBER
- DF = 0 DENOTES A BORROW. D IS TWO'S COMPLEMENT

THE SYNTAX "−(NOT DF)" DENOTES THE SUBTRACTION OF THE BORROW

Appendix F—
ASCII-Hex Table

MOST SIGNIFICANT HEX DIGIT

LEAST SIGNIFICANT HEX DIGIT	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	¦
D	CR	GS	—	=	M	]	m	}
E	SO	RS	.	>	N	↑	n	~
F	SI	US	/	?	O	←	o	DEL

NOTES:

(1) Parity bit in most significant hex digit not included.

(2) Characters in columns 0 and 1 (as well as SP and DEL) are non-printing.

(3) Model 33 Teletypewriter prints codes in columns 6 and 7 as if they were column 4 and 5 codes.

92CS-34736

Appendix G—
Logic Diagram, Layout Diagram, and Parts List for CDP18S652 Microboard Combination Memory and Tape I/O Control Module

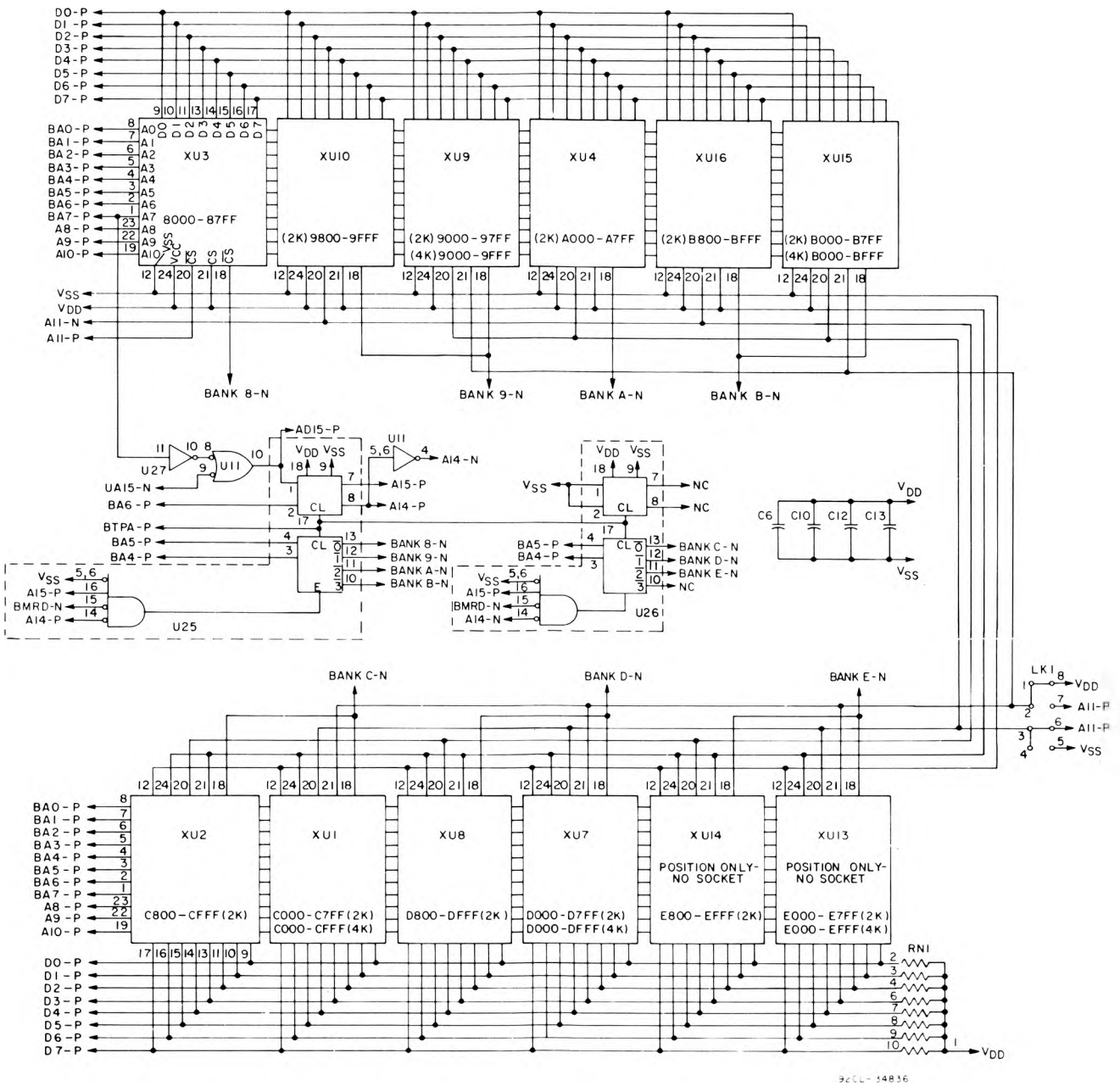


Fig. E1 - Logic diagram of CDP18S652 - memory portion.

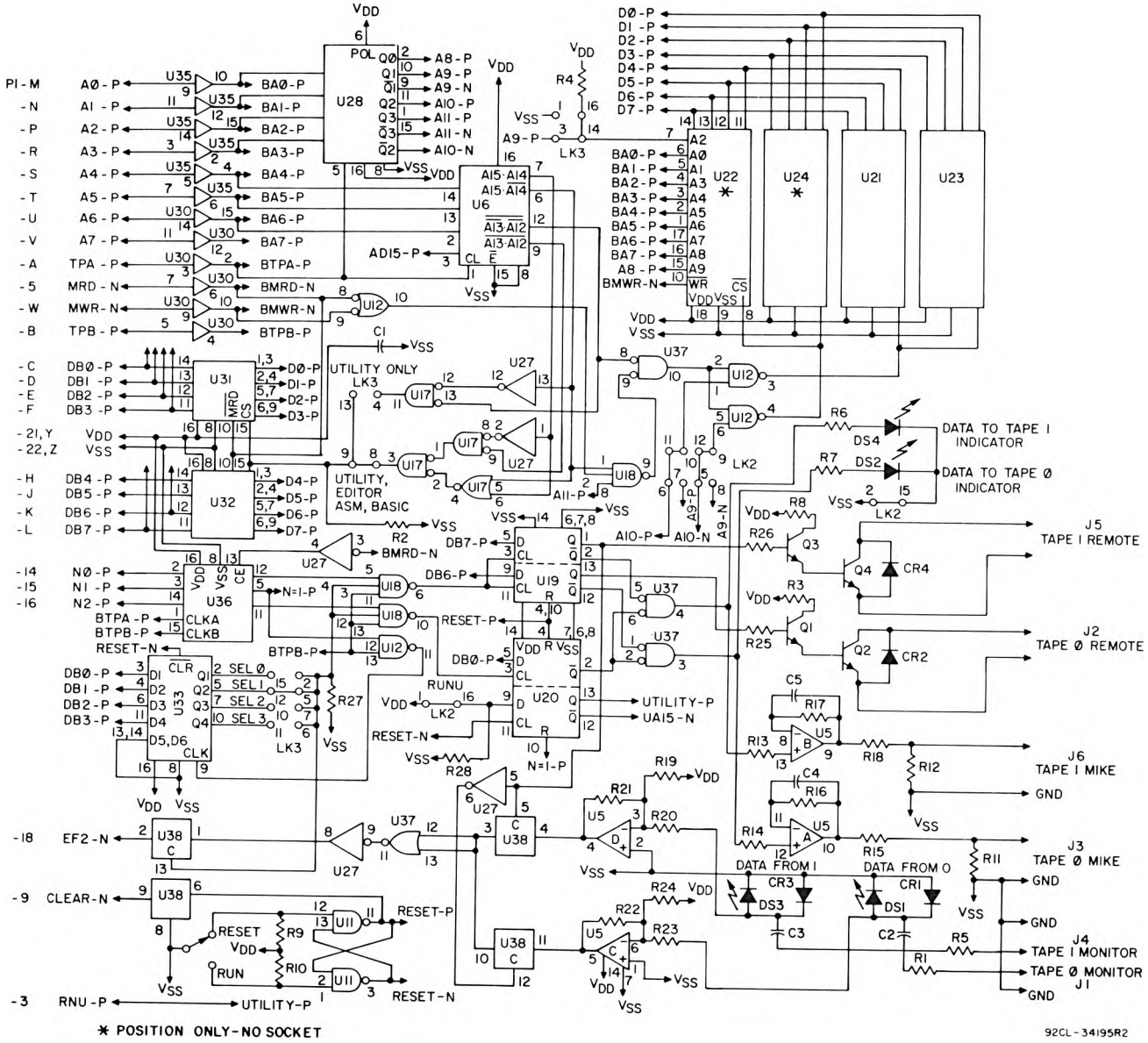


Fig. G2—Logic diagram of CDP18S652—control and RAM portion.

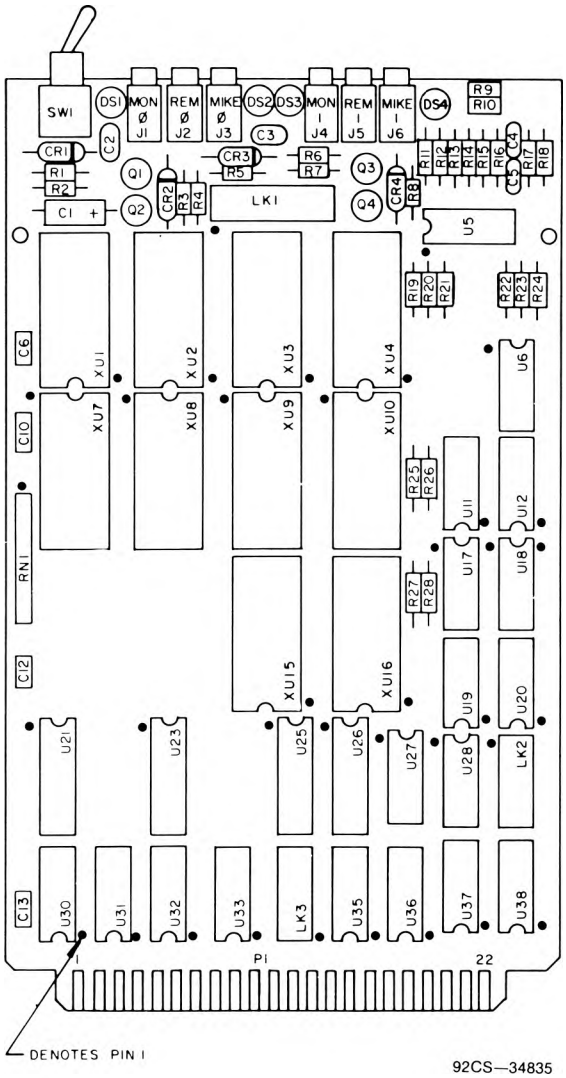


Fig. G3—Layout diagram of CDP18S652.

Parts List

C1=22μF, 16 V
C4, C5=0.0047 μF, 50 V
C2, C3, C9, C11=0.47 μF, 50 V
C6, C10, C12, C13=0.1 μF

CR1-CR4=1N914

DS1-DS4=LED HP5082-4494

J1-J6=phone jacks

R1, R5, R11, R12=100 ohms, 0.25W
R6, R7 =680 ohms, 0.25 W
R3, R8, R15, R18=1 kilohm, 0.25 W
R25, R26=5.1 kilohms, 0.25 W
R16, R17=10 kilohms, 0.25 W
R13, R14=100 kilohms, 0.25 W
R19, R21, R22, R24=1 megohm, 0.25 W
R2, R4, R9, R10, R20, R23, R27, R28=
22 kilohms, 0.25 W

RN1=resistor network, 10 kilohms

*U1, U2, U7, U8, U15, U16=BASIC3 EPROM's
*U3=UT62 EPROM
¢U4, U9, U10=Editor/Assembler EPROM
U5=CA3401E
U6=CDP1858CE
U11, U12=CD4011UBE
U17,U37=CD4001UBE
U18=CD4023BE
U19.U20=CD4013BE
U21,U23=RAM 1024× 4
U25,U26=CDP1866CE
U27=CD4069UB
U28=CD4042BE
U30, U35=CD4050BE
U31, U32=CDP1856CE
U33=CD4017BE
U36=CDP1853CD
U38=CD4066BE

*Supplied with CDP18S693 and CDP18S694

¢Supplied with CDP18S694 only

EPROM Locations

EPROM	SOCKET
BASIC 1	U15
BASIC 2	U16
BASIC 3	U1
BASIC 4	U2
BASIC 5	U7
BASIC 6	U8
EDIT-ASM 1	U9
EDIT-ASM 2	U10
EDIT-ASM 3	U4
UT62	U3

Appendix H—
Hardware Description for PROM Programmer Module

Fig. H1 shows a block diagram of the programmer module. It consists of two output ports that latch and hold addresses for the PROM's, an input and output port connected to the PROM data bus, and two more output ports that supply control signals such as chip select and programming levels to the PROM's. All internal signals operate at 5 volts as is common to the MCDS. Discrete transistor drivers operated by control signals are used to switch higher voltages to the PROM's.

Group-Select Logic

Fig. H2 gives a block diagram of the group-selection logic on the PROM Programmer module. The execution of output instruction '61' generates a clock input to the flip-flop. The D-input of that flip-flop is connected by a link to one of nineteen possible signals. The high-order four bits of the data bus are decoded into fifteen lines (the decode of all zero is not used); the low-order four bits are supplied directly. Thus, a total of nineteen unique addresses is available to be assigned to the PROM Programmer Module. The Module comes with the link installed in position C, connecting data bus bit DB2 as the enabling signal. The Module is thus enabled by output instruction '61' with data byte=00000100.

Power Supply Control

The supply voltages to the PROM's are sent through the Power ON/OFF switch S2 and the software sets

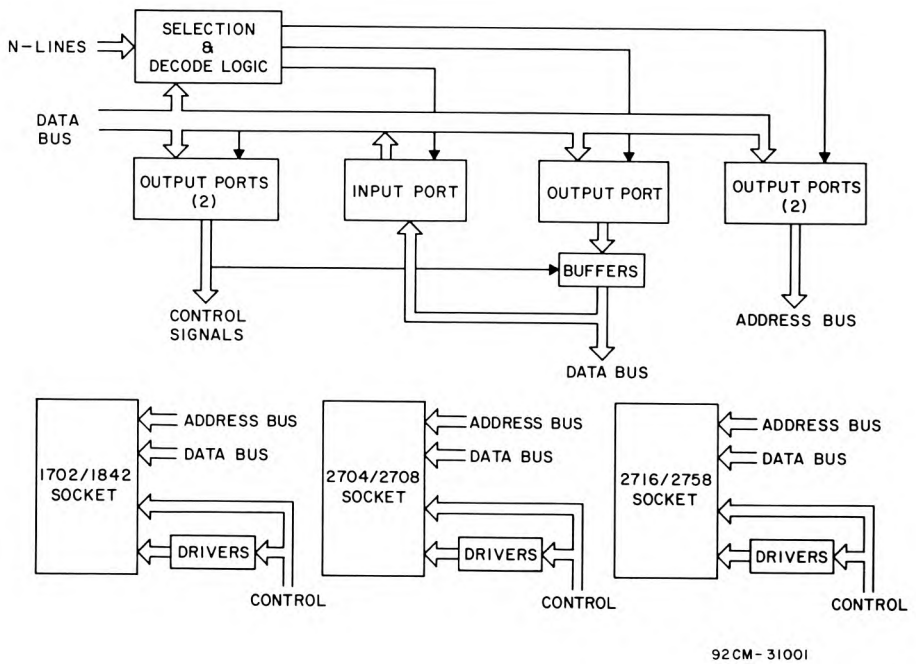


Fig. H1—Block diagram of PROM programmer module.

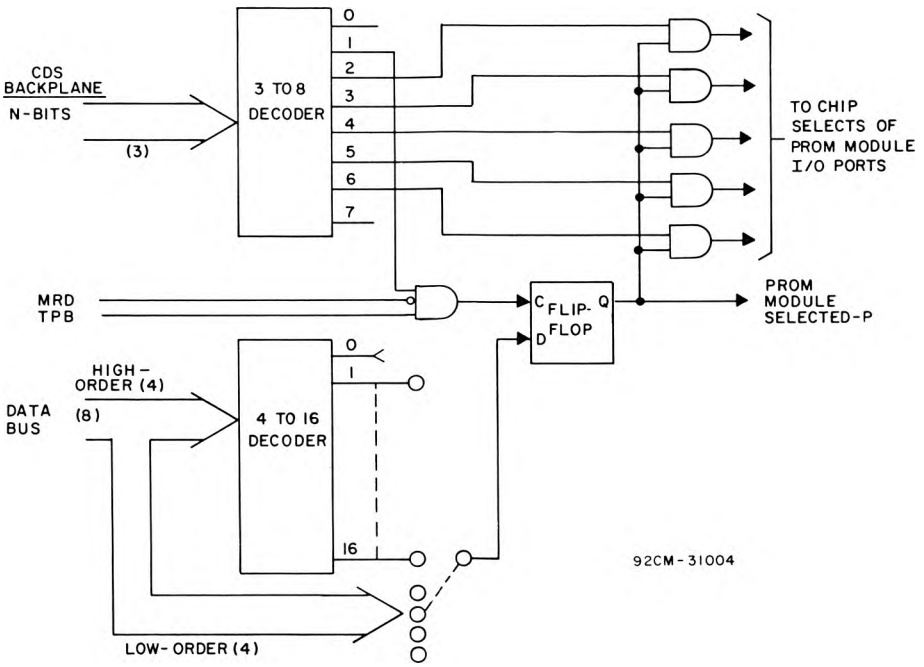


Fig. H2—Block diagram of PROM programmer module group-selection logic.

address, data, and control signals at ground after each operation so that PROM's can be inserted and removed without having power applied to them. The I/O ports and all logic circuits on the board are continuously powered by the +5-volt Vcc supply from the MCDS. A power-up reset circuit is included to assure that no unwanted voltages will be applied to a PROM should one accidentally be in a socket when the MCDS is turned on.

A selection switch S1 selects either the 1702 or the CDP18U42 configurations for socket 1. The switch generates a signal to EF1 of the CPU which is tested by the software at the start of an operation to make sure the switch is in the correct position before proceeding. If some other piece of user-installed hardware, not covered by a group select, is connected to EF1, some alternative arrangement will have to be made.

Three LED status indicators are provided on the edge of the Module. The one labeled PROG will be ON when a PROM is actually being programmed and OFF at other times. The one labeled 5 V indicates that +5 V is being applied to the PROM's. This indicator is controlled by power switch S2. PROM's should not be inserted or removed when this LED is ON. The third LED, labeled PROG PWR, indicates that the external programming voltage power supply is ON and connected through to the drivers circuits on the Module.

Schematics, Layout Diagram, and Parts List

Figs. H3, H4, H5, and H6 contain the schematic and logic diagrams for the PROM Programmer Module. Fig. H7 is the layout diagram and is followed by the parts list.

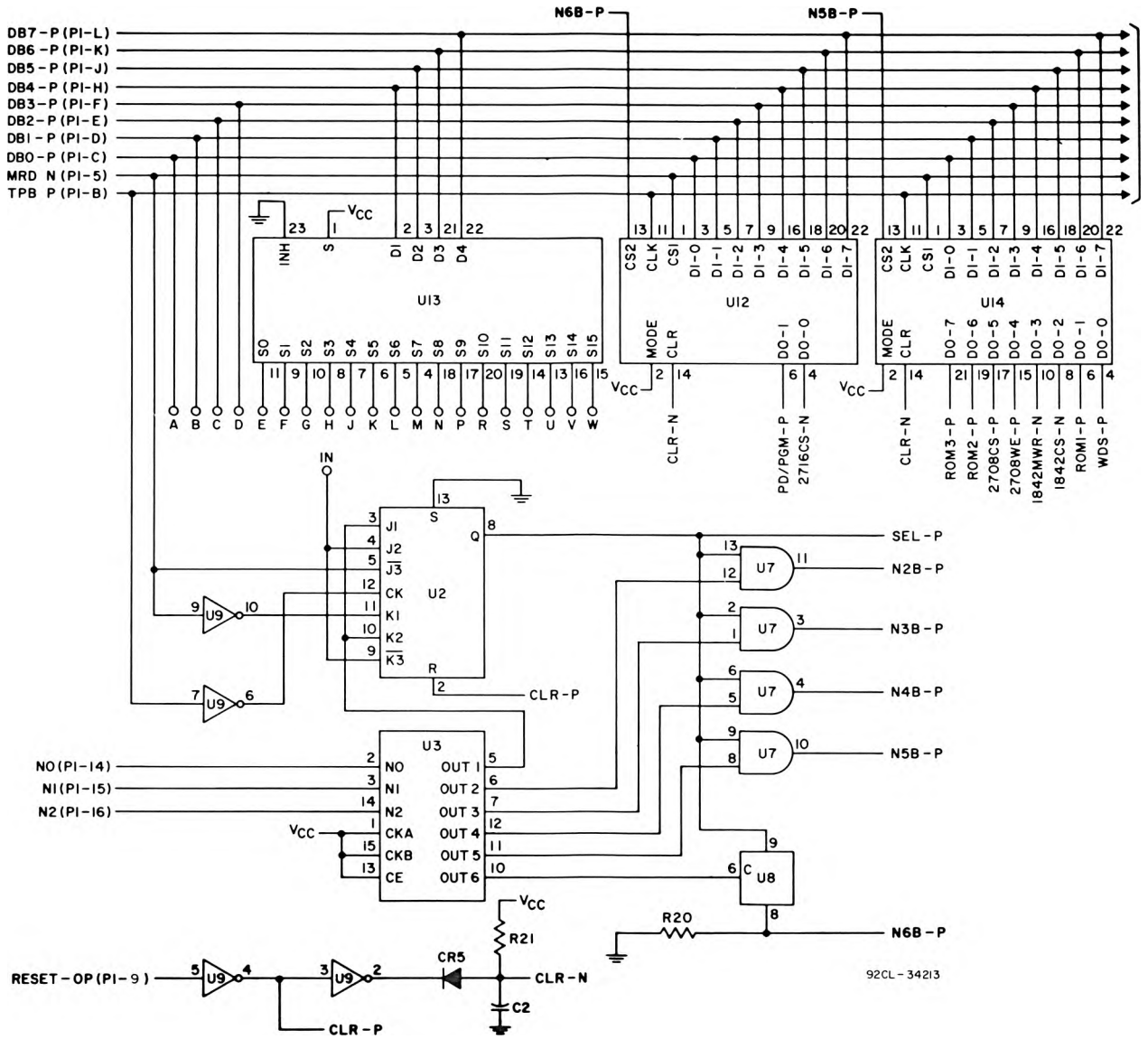


Fig. H3—Control logic of PROM programmer module.

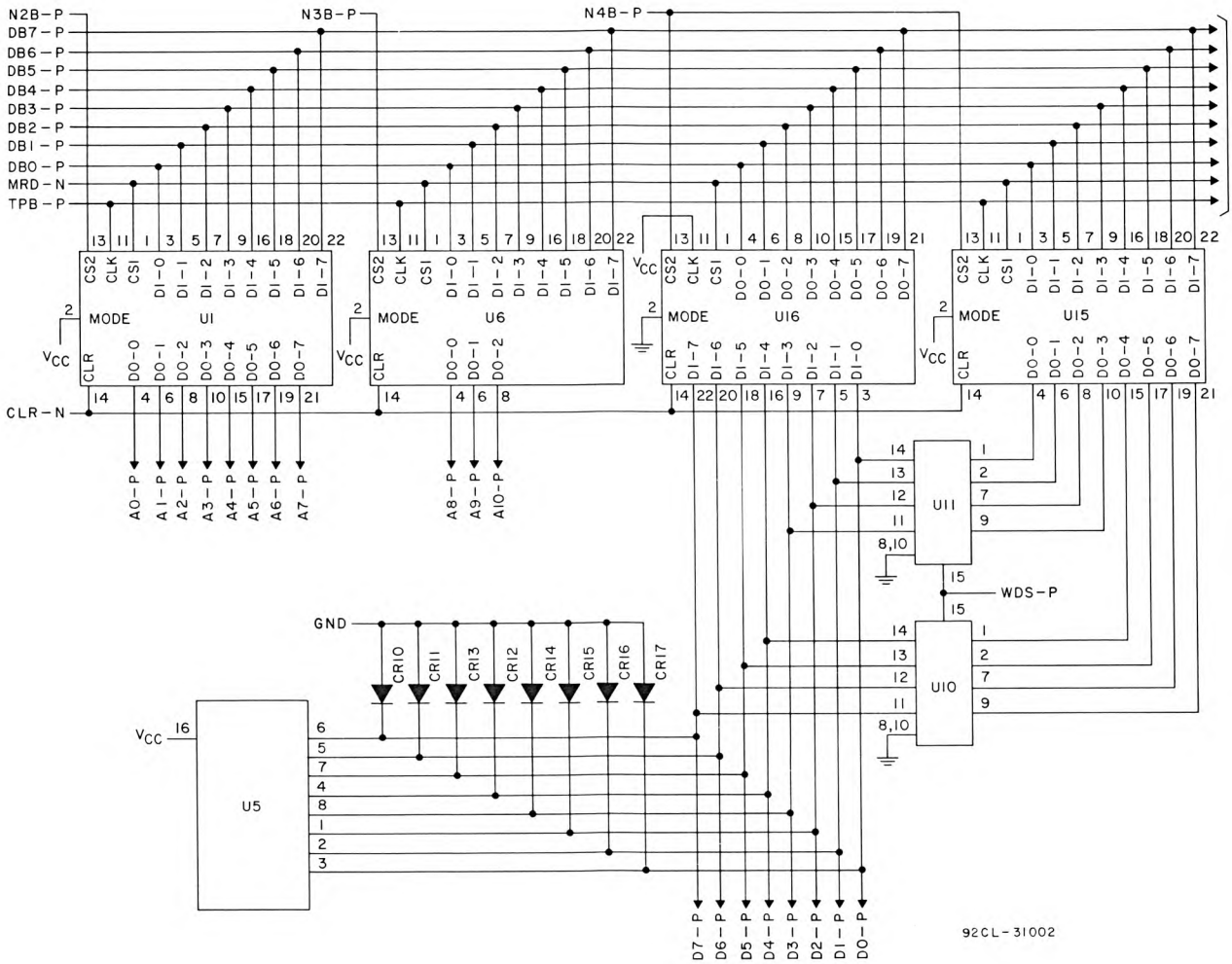


Fig. H4—Address and data logic of PROM programmer module.

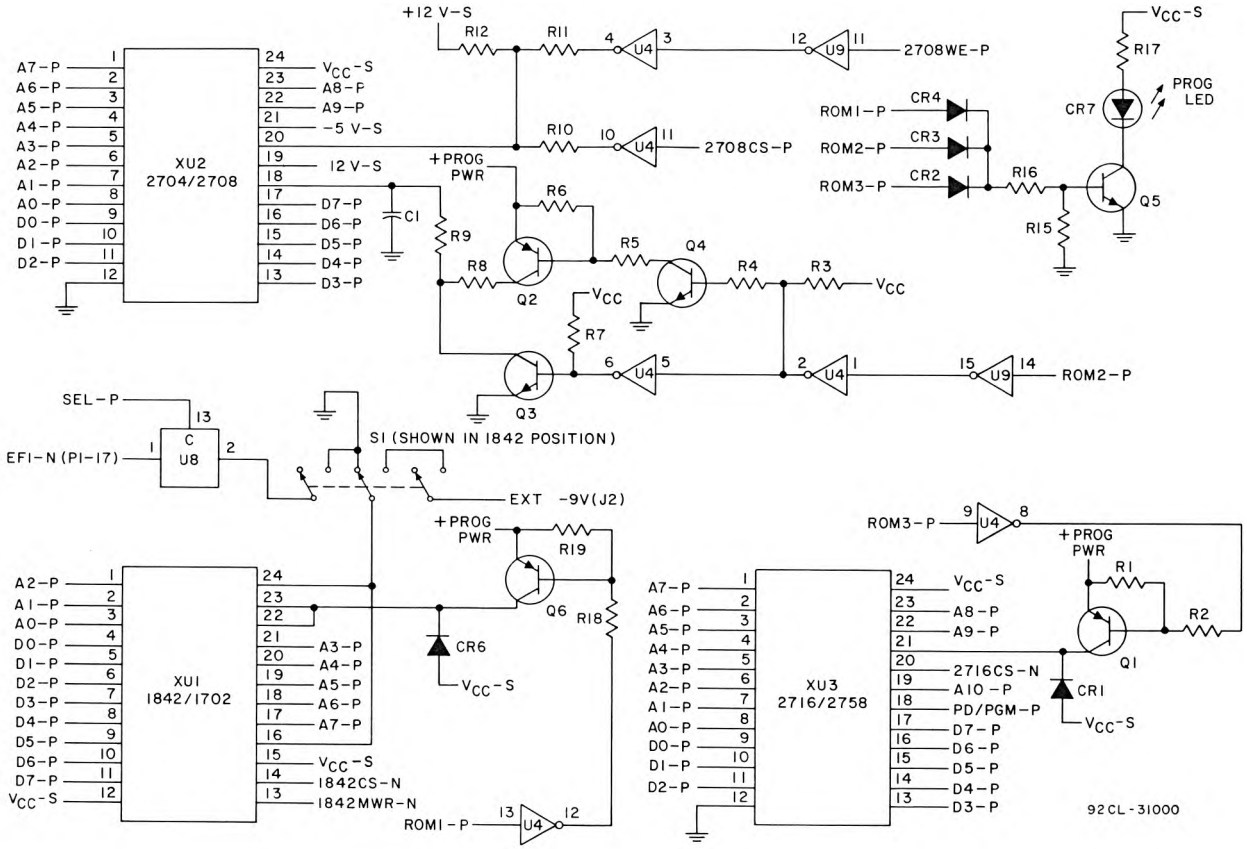


Fig. H5—Programming logic of PROM programmer module.

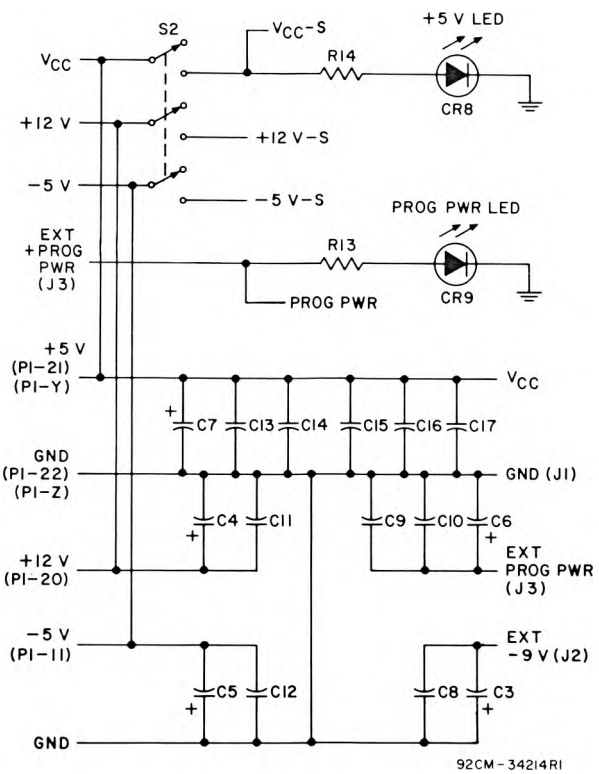


Fig. H6—Power supply of PROM programmer module.

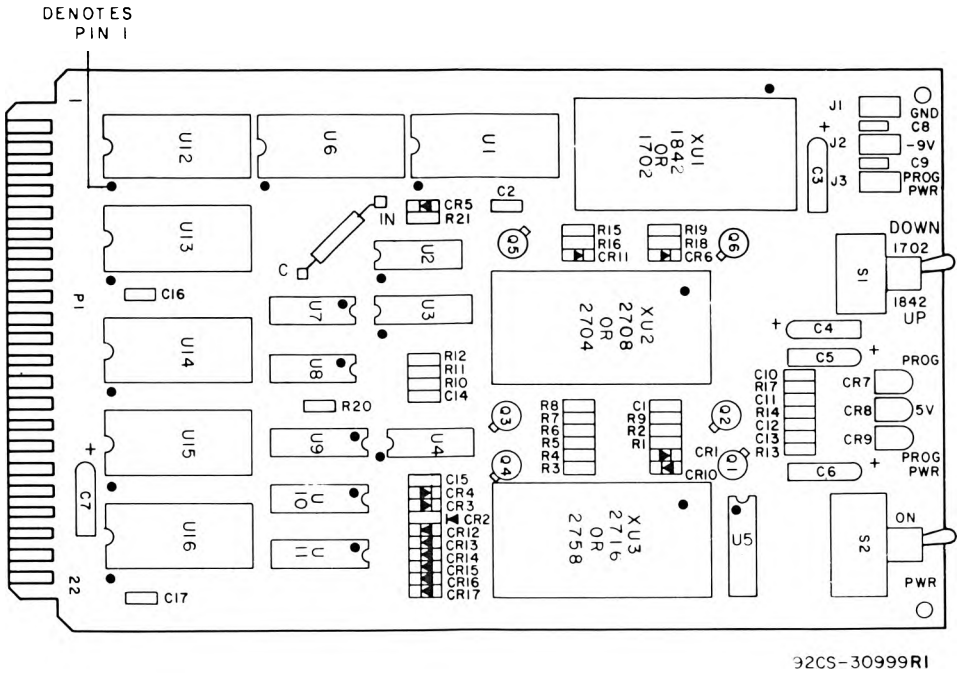


Fig. H7—Layout diagram of PROM programmer module.

Parts List

C1, C2=0.0022 μF
C3, C4, C5, C7=15 μF, 20 V
C6=15 μF, 50 V
C8, C9, C10, C11, C12, C13, C14, C15, C16, C17=0.1 μF, 50 V
CR1, CR6=1N34A
CR2, CR3, CR4, CR5, CR10, CR11, CR12 CR13, CR14, CR15, CR16, CR17=1N914
CR7, CR8, CR9=LED red

J1=jack, black
J2=jack, green
J3=jack, yellow

Q1, Q2, Q6=2N2907
Q3, Q4, Q5=2N2222

R1, R6, R19, R20=10 kilohms, ±5%, 0.25 W
R2, R4, R5, R18=3000 ohms, ±5%, 0.25 W
R3=1000 ohms, ±5%, 0.25 W
R7, R10, R16=4700 ohms, ±5%, 0.25 W
R8=47 ohms, ±5%, 0.25 W
R9, R11=100 ohms, ±5%, 0.25 W
R12, R21=6800 ohms, ±5%, 0.25 W
R13=910 ohms, ±5%, 0.25 W
R14, R17=200 ohms, ±5%, 0.25 W
S1=3PDT switch
S2=3PDT switch

U1, U6, U12, U14, U15, U16=CDP1852E
U2=CD4097BE
U3=CDP1853E
U4=SN7406N
U5=resistor network, 22 kilohms
U7=CD4081BE
U8=CD4066BE
U9=CD4049BE
U10, U11=CDP1856E
U13=CD4514BE

XU1, XU2, XU3=24-terminal socket

Appendix I—
PROM Programmer Specifications

Basic Operations:

- Program a PROM from a RAM buffer or file; automatically followed by a verification
- Verify a PROM against RAM buffer or file
- Copy a PROM into RAM buffer, automatically followed by a verification
- Fill RAM buffer with all 1's or 0's; used in verifying PROM erasure
- Save RAM buffer onto a file

Operating Temperature Range:
0° to 43°C

Plug-In Module
Dimensions: 4.5 × 7.5 inches (114.3 × 190.5 mm)
Three Zero-Insertion Force PROM Sockets
1 for 1702/CDP18U42
1 for 2704/2708
1 for 2716/2758
Plugs into Microboard chassis
Assigned to Group Select 4

Power Supplies:
External-Programming Power:
See Manufacturer's data specifications sheet

LED Indications:
Power ON to PROM
External Programming Power ON
Programming ON

Switches:
Power to PROM ON/OFF
Selector Switch

Programming Times:
2708 - 2 minutes 45 seconds
2716 - 1 minute 45 seconds
2758 - 50 seconds
CDP18U42 - 3 seconds

Types of PROM's Handled:

1702	256 word by 8 bit - read only	Intel PROM's or equivalent
2704	512 word by 8 bit	
2708	1024 word by 8 bit	
2758	1024 word by 8 bit	
2716	2048 word by 8 bit - single voltage only	
CDP18U42	256 word by 8 bit	

Appendix J—
Generating Hard Copy on MCDS.

The MCDS can produce hard-copy listings of user code from cassette tape with a hard-copy terminal, a low-cost serial-interface printer, or a printer having a parallel Centronics-type interface.

If a hard-copy terminal is used, the Editor can dump a tape into the terminal with the following command sequence:

(N)<AAB100T100K>(ESC-ESC)

This command string tells the Editor to read in 100 lines, print them with the "T" command, and then delete 100 lines. The sequence is repeated until "N" times 100 lines are read or end of file is encountered.

This same sequence can be used with a video terminal and a low-cost serial printer in combination. The CDP18S601 Microboard Computer has both an RS-232C and a 20-mA loop connector. The serial printer can gets its input from the alternate (unused) terminal connector.

For parallel printer application, the CDP18S646 Microboard printer interface can be installed in the system and the following Editor command typed:

(N)<AAB100P100K>(ESC-ESC)

This sequence operates like the serial-printer sequence except that the "P" command directs the Editor to output data to the parallel interface port of the CDP18S646.

BASIC3 programs stored on tape with the "PSAVE" command cannot be listed because they are saved in "pseudo-code" and are not ASCII. BASIC3 ASCII listings can be saved on tape and hard-copied by performing a "LIST" command to tape. Refer to the manual **BASIC3 High-Level-Language Interpreter CDP18S841 User Manual**, MPM-841, for detailed operation of "LIST" and "DOUT" commands.

Index

A commands 9
Aborting PROM programming operations 53
Adding to a file 26
Address constants—CRA 31
Apostrophe 32
Application hints—PROM programmer 57
ASCII coding 9
ASCII to Hex conversion routine 13
ASCII-hex table 92
Assembly session example 28

B commands 9
BASIC3 16
BASIC3 cold start 16
BASIC3 warm start 16
Bidirectional mode 46
Binary constants—CRA 30
Bit-programmable mode 46
Block diagram of CDP18S694 system 37
Branch addresses—CRA 31
Buffer 18
Buffer pointer 19

C commands 9
C command—PROM programmer 56
Cable CDP18S515 5
Cable CDP18S516 5
Cable connections for system 6
Cabling 5
Cassette recorder unit 49
Cassette tape drive 49
Cassette tape interface cable 6
Cassette tape unit operation 49
CDP1802 instruction summary 89
CDP1804 instruction summary 89
CDP1805 instruction summary 89
CDP1851 programmable I/O interface 45
CDP18S601 Microboard computer 37
CDP18S652 combination memory and tape I/O module 38
CDP18S652 logic diagrams and parts list 94
Character sequence 18
Cold start—Basic 3 16
Combination memory and tape I/O module CDP18S652 38
Command formats 20
Command nesting—CRE 23
Command strings 20
Comments—CRA 30
Composite commands—CRE 23
Compound operations—PROM programmer 58
Connection list for interface cables 88
Constants, address—CRA 31
Constants, binary—CRA 30
Constants, decimal—CRA 30
Constants, explicit—CRA 30
Constants, expression—CRA 30
Constants, hexadecimal—CRA 30
Constants, text—CRA 30
Controls on tape unit 49
Copy operations—C command 56
Correcting PROM programmer typing errors 53
Correcting typing errors—CRE 21
COSMAC Resident Assembler—CRA 28
CRA directives 32
CRA error messages 34
CRA sample program 33
CRA sample program symbol table 33
CRE command operation 21
CRE command summary 23
CRE commands—single 21
CRE commands 21
CRE composite commands 23
CRE control character summary 23
CRE COSMAC Resident Editor 17
CRE data manipulation examples 27
CRE error messages 24
CRE execution interruption 21
CRE line length 19
CRE memory space requirements 18
Creating a file 25
Custom I/O modules 46
Custom memory modules 42

D commands 7
Data format for cassette unit 49
Data lists 32
Data manipulation commands—CRE 22
Data synchronization 45
Data terminal 5
Decimal constants—CRA 30
Deleting a file section 26
Deletion commands—CRE 22
Diagrams and part list for PROM programmer 97
Diagrams and parts list for CDP18S652 94
Directives—CRA 32
DMA input 47
DMA output 47

E command 17
E command 9
Editor 17
END statement 32
Error correction—CRE 21
Error messages—CRA 34
Error messages—CRE 24
Error messages—PROM programmer 53
Examples of compound operations—PROM programmer 58

Examples of UT62 usage 12
Expansion capabilities 41
Explicit constants—CRA 30
Expression constants—CRA 30
Expression evaluation—CRA 29
External flags EF1 to EF4 47

F command—PROM programmer 56
F commands 8
File creation 25
File development 25
File manipulation 25
File reader subroutines—PROM programmer 57
Files 18
Fill command—PROM programmer 56
Filled work space warning 21

Group select logic—PROM programmer 97

Hard copy generation 104
Hardware description—PROM programmer 97
Hardware structure 37
Hexadecimal constants—CRA 30
Horizontal tabs 25

I commands 7
I/O device operands—CRA 31
I/O expansion 46
I/O interfacing, two leel 44
I/O modules available 46
Immediate operands—CRA 31
Initialization routines 13
Input files 18
Input/Output interfacing 44
Installation of PROM programmer 51
Instruction summary for microprocessors 89
Instructions—CRA 31
Interface cable connection list 88
Interfacing parallel I/O 45
Interfacing, serial I/O 44
Interrupt 47
INTERRUPT lines 46
Interrupting CRE execution 21

J command—PROM programmer 56
J commands 8

Layout diagram for CDP18S652 96
Layout diagram for PROM programmer 102
Level I assembly language 28
Level I assembly language 30
Line length restrictions 19
Lines—CRA 30
Listing, monitor program UT62 60
Location counter—CRA 29

M commands 8
MCDS specifications 59
Memory addressing 42
Memory buffer format 19
Memory expansion 42
Memory FILL routine example 13
Memory map—PROM programmer 52
Memory map 42
Memory modules available 42
Memory MOVE routine example 42
Memory MOVE routine example 13
Memory organization—PROM programmer 51
Microboard computer CDP18S601 37
Microboard Computer Development
System specifications 59
Mnemonics—CRA 31
Modifying a file section 26
Module descriptions 37
Monitor program listing—UT62 60
Monitor program operation test 6
Monitor program UT62 7
Moving a file section 26
Multiple interrupts 47

n interpretations 20
Nesting of commands 23

Operand, I/O devices—CRA 31
Operands—CRA 31
Operands, immediate—CRA 31
Operands, register—CRA 31
Operation memonics—CRA 31
Operation of cassette drive unit 49
Operations and commands—PROM programmer 54
ORG statement 32
OSTRNG routine example 13
Output commands—CRE 22
Output files 18

P command—PROM programmer 54
P commands 8
PAGE statement 32
Parallel I/O interfacing 45
Parity bit 9
Parts list for CDP18S652 96
Parts list for PROM programmer 102
Pin terminals—universal backplane 38
Pointer 19
Pointer control commands 21
Power supply control—PROM programmer 97
Power-on reset 48
Printing hard copy 104
Program flow chart 18
Programmable I/O interface CDP1851 45
Programming operations—P command 54
Programming voltages 51
PROM programmer application hints 57
PROM programmer compound operations 58
PROM programmer error messages 53
PROM programmer module 51
PROM programmer module hardware description 97
PROM programmer operation precautions 53

PROM programmer programming voltages 51
PROM programmer software memory map 52
PROM programmer software 52
PROM programmer usage 51
Pseudocode 16

R commands 8
RAM address 41
RAM table for READ/WRITE routines 15
RCA I/O modules 46
RCA memory modules 42
Read and type routines 9
READ routine example 12
Read routines—UT62 9
READ TAPE routines 14
Reading input tape commands 21
READY lines 45
Record formats 18
Register operands—CRA 31
Register utilization—UT62 9
Register save 7
ROM address 41
Running Basic 3 16
Running the assembler—CRA 28

S command—PROM programmer 56
S commands 8
Sample assembly session 28
SAVE operation—S command 56
Serial I/O interfacing 44
Setting up the system 5
Signals—universal backplane 38
Software—PROM programmer 52
Spaces 32
Specifications for MCDS 59
Starting the system 5
Subroutine calling sequence 11
Summary of CRE commands and control characters 23
Symbol definitions—CRA 30
Symbol table—CRA 29
Symbol table for sample program 33
System cable connections 5
System care 49
System operation tests 6
System setup 5
System signals—I/O interfacing 47
System signals 39, 40
System startup 5

Tape considerations 49
Tape file format 19
Tape handling precautions 87
Tape load 7
Tape READ and WRITE routines 14
Tape READ routines 14
Tape recommendations 49
Tape save 7
Tape TYPE routines 14
Terminal cable 5
Terminal interface cable connection list 88
Terminal, power requirements 5
Text constants—CRA 31
Text editor 17
Text insertion commands—CRE 22
TIMALC subroutine 10
Timing parameter TP 10
Two-level I/O interfacing 44
Type routines—UT62 12
TYPE routines examples 12

U command—PROM programmer 56
Universal backplane 38
Using CRE 25
Using PROM programmer 51
Using the assembler 29
UT62 commands 7
UT62 monitor program 7
UT62 monitor program listing 60
UT62 reentry 14
UT62 register utilization 11
UT62 usage example 12

V command—PROM programmer 55
Vectored interrupts 48
Verify operations—V command 55

W commands 8
Warm start—BASIC3 16
WRITE TAPE routine 14

FCC Compliance Report

The RCA Microboard Computer Development Systems CDP18S693 and CDP18S694, tested as subsets of the CDP18S695, comply with the FCC Rules, Part 15, Subpart 3 for Class A Computing Devices. The verified test report is given below.

Radiated Emissions

Frequency	Reading	Antenna Factor	μV/m at 3 Meters
47	80	9	65
48	80	9	65
49	80	9	65
50	80	9	65
79	80	11	80
86	80	11	80

Line Conducted Emissions

	Side A		Side B	
Frequency	Reading	Signal Level	Reading	Signal Level
0.45 kHz	-60	220	-60	220
1.5 MHz	-60	220	-60	220
29.8 MHz	-60	220	-56	355